修士論文

MPICH-Vclに対する
チェックポイントサーバー冗長化の手法

電気通信大学　大学院情報システム学研究科
情報システム基盤学専攻

0753021　　増永 明剛

指導教員　　本多弘樹
古賀久志
近藤正章

2009年 1月 29日 提出

目 次

図 目 次

表 目 次

1 はじめに

1.1 研究の背景

コンピュータクラスタは多数の計算機をネットワークに接続し，一つの並列計算機システムにしたものである．市場に多く出回っているパーツを使用しているため，比較的安価に高性能な計算能力を持ったシステムを構築できる．一方で安価なパーツが用いられていることや，多数の計算機からシステムが構成されているために，システム全体としての信頼性が低いという問題点もある．

コンピュータクラスタにおける故障としてはハードディスク等のパーツの故障や，計算機ノード間の通信エラーなどがあげられる．コンピュータクラスタで実行される科学技術計算などのアプリケーションは，実行に数時間から数日，数ヶ月かかることもある．そのためアプリケーションの実行中に故障が発生する確率が高い．アプリケーションの実行中に故障が発生した場合，故障に対する対策が行われていなければ計算の途中結果は失われ，アプリケーションを最初から実行しなおさなければならない．そこで，コンピュータクラスタは故障が発生してもアプリケーションの実行を続けることができるように，耐故障性を備えていることが望ましい．

コンピュータクラスタで実行される科学技術計算のアプリケーションで用いられる並列プログラミングの規格としてはMPI（Message Passing Interface）[1]が普及している．MPIの機能はライブラリとして提供される．MPIの実装としてはOpen MPI[2]，LAM/MPI[3]，MPICH[4]，MVAPICH[5]などが知られている．

MPIに耐故障性を付加する研究はこれまでにも行われてきた．MPIにおける耐故障性を付加する方式の1つとして，ロールバックリカバリプロトコル[9]を用いる方式が提案されている．この方式は，実行中のMPIアプリケーションの途中結果（チェックポイント）をチェックポイントデータとして保存（チェックポインティング）し，故障が発生した場合，チェックポイントデータを用いてチェックポイントからMPIアプリケーションの再実行（ロールバックリカバリ）を行うものである．

一方，MPIアプリケーションに耐故障MPIライブラリを適用する方式は以下の2つに大別できる．

1つは，耐故障専用のオリジナルライブラリを提供し，プログラマがMPIプログラム中でそのライブラリを呼び出す形で利用する方式である．この方式はアプリケーションにあわせた耐故障方式を選択することが可能である．そのため耐故障性を付加することによるオーバーヘッドを低く抑え，利用可能な計算機ノードを有効に利用できる．しかし，プログラマは故障を考慮してアプリケーションを作成しなけれ

ばならない．また，既存の MPI アプリケーションに耐故障性を付加するためにはプログラムを変更しなければならないため，プログラマにかかる負担が大きい．

もう 1 つは，既存の MPI ライブラリに耐故障性の機能を組み込む方式である．この方式では，前述の方式のように耐故障性を付加するためにプログラムの変更は不要であるが，アプリケーションにあわせて適切な耐故障方式を選択することが難しく，耐故障性を付加することによるオーバーヘッドが大きくなる．また，正常な計算機ノードを効率的に利用することも難しい．

上記に述べた 2 つの方式の内，MPI ライブラリに耐故障性の機能を組み込む方式を実装した耐故障 MPI ライブラリとして MPICH-Vcl[11, 14] が知られている．MPICH-Vcl は MPI の実装の一つである MPICH[4] に対して，ロールバックリカバリプロトコルの一つである Coordinated Checkpointing[9] を実装した耐故障 MPI ライブラリである．MPI アプリケーションに対して耐故障性を付加するために，MPICH-Vcl では MPI アプリケーションに対して耐故障性を付加する専用のプロセスを導入している．MPICH-Vcl では耐故障性に関する機能は MPI ライブラリに組み込まれているので，耐故障性を付加するためにかかるプログラマへの負担は少ない．一方で，MPICH-Vcl には MPI アプリケーションに対して耐故障性を付加するために導入された管理プロセスには冗長性がない．そのため，管理プロセスに故障が発生した場合，故障からのロールバックリカバリができず，MPI アプリケーションの実行が停止してしまう．このことは MPICH-Vcl を MPI アプリケーションに対して耐故障性を付加するツールとして利用する上で問題となる．

1.2 研究の目的

本研究では MPICH-Vcl の耐故障性を向上させることを目的とする．そのために，MPICH-Vcl の管理プロセスの 1 つである，チェックポイントサーバーに対して故障対策を行う．チェックポイントサーバーはチェックポイントの管理を行うプロセスである．チェックポイントサーバーに故障が発生するとチェックポインティング時に MPI アプリケーションの実行が停止する．また，MPI プロセスに故障が発生した場合にロールバックリカバリが不可能になる．そこで，チェックポイントサーバーに対して故障対策が必要になる．チェックポイントの冗長性を維持できるため，チェックポイントサーバーにおける故障対策として冗長化を考える．しかし，単純にチェックポイントサーバーを冗長化したのではチェックポインティングによるオーバーヘッドが増してしまうことが予想されるため，チェックポイントにおける耐故障の手法を参考にチェックポイントサーバー冗長化の手法を提案する．提案方式におけるチェッ

クポイントサーバー冗長化によるチェックポインティングオーバーヘッドや対応可能な故障のパターンなどの特性を MPICH-Vcl と比較することにより，本手法の有効性を検証する．

1.3 論文の構成

2 章ではこれまでに提案された MPI におけるロールバック・リカバリプロトコルについて述べる．3 章では複数のノードでの故障発生に対応可能なチェックポインティングプロトコルについて述べる．4 章は耐故障 MPI ライブラリ MPICH-Vcl の実装について述べる．5 章は MPICH-Vcl におけるチェックポインティングによるオーバーヘッドと，対応可能な故障について調査した結果をまとめ，調査結果から検討した MPICH-Vcl の問題点を述べる．6 章は MPICH-Vcl の問題点に対する対策の検討と，チェックポイントサーバー冗長化の方式を検討した結果について述べる．7 章はチェックポイントサーバー冗長化のための具体的な実装方式を説明する．8 章は MPICH-Vcl に対してチェックポイントサーバー冗長化の実装を行った場合の動作検証とチェックポインティングによるオーバーヘッドの測定結果をまとめ，考察を行う．9 章では関連研究と本研究との研究対象の違いを述べる．10 章で結論と今後の課題について述べる．

2 MPIにおけるロールバックリカバリプロトコル

MPI（Message Passing Interface）におけるロールバックリカバリプロトコル [9] は，MPI アプリケーションの計算の途中結果をチェックポイントとして保存し，故障が発生した場合はチェックポイントから再実行（ロールバックリカバリ）する耐故障方式である．

MPI におけるロールバックリカバリプロトコルは，MPI プロセスの途中結果を保存するチェックポインティングをベースとした Checkpoint-Based Rollback-Recovery と，チェックポイントのほかに保存した MPI プロセス間の通信などの非決定イベントのログを，ロールバックリカバリに使用する Log-Based Rollback-Recovery の 2 種類に分けられる．

本章では Chekcpoint-Based Rollback-Recobery と Log-Based Rollback-Recobery の 2 つのロールバックリカバリプロトコルについて，文献 [9] に従って述べる．

2.1 Checkpoint-Based Rollback-Recobery

Coordinated Checkpointing はプロセスをロールバックリカバリする際にチェックポイントのみを用いるロールバックリカバリプロトコルである．チェックポインティング時に MPI プロセス間で同期を取る Coordinated Checkpointing，チェックポインティング時に MPI プロセス間の同期を取らない Uncoordinated Checkpointing，各プロセスがある特定の条件の下で同期を行わずにチェックポインティングを行う Communication-induced Checkpointing の 3 種類がある．Checkpoint-Based Rollback-Recovery はロールバックリカバリ時に一貫性のあるチェックポイントが必要になる．そのため，1 つでもチェックポイントが欠けているプロセスが存在し，チェックポイントの一貫性がとれない場合，ロールバックリカバリできない．以下，それぞれについて詳述する．

2.1.1 Coordinated Checkpointing

チェックポインティング時に各プロセスが同期を取り，通信の一貫性を保障したチェックポイントを取る．この方式は同期のためのオーバーヘッドが大きい．しかし，全てのチェックポイントが一貫性を保持しているため，無駄なチェックポイントを作る必要がなく，古いチェックポイントは破棄することができる．同期の方式により Blocking と Non-Blocking の 2 種類に分けられる．Blocking 方式はチェックポインティング時に MPI プロセスを停止させ，チェックポインティングの処理のみを行

う．Non-Blocking 方式は MPI プロセスとチェックポンティングを同時並行に行う．MPICH-Vcl は Non-Blocking Coordinated Checkpionting 方式を実装している．

2.1.2 Uncoordinated Checkpointing

各プロセスが任意にチェックポインティングを行い，ロールバックリカバリ時に通信の一貫性が保たれるチェックポイントを選ぶ．

この方式では同期を行わないため，同期によるオーバーヘッドがない．しかしながら最後の一貫性の取れるチェックポインティング以降にとられたチェックポイントは，全て無駄になってしまう（ドミノ効果）．

2.1.3 Communication-induced Checkpointing

各プロセスがある特定の条件の下にチェックポインティングを行う．これにより同期を行わず，さらにドミノ効果の発生を防ぐこともできる．

このような条件は複数知られている．MRS model[10] と呼ばれる条件は，あらゆる受信があらゆる送信よりも先に起こればドミノ効果が起こらないというもので，全ての受信の直前にチェックポインティングを行う．

2.2 Log-Based Rollback-Recovery

Log-Based Rollback-Recobery は，チェックポインティングに加えて，プロセス間の通信などの非決定イベントのログを保存し，これを利用するロールバックリカバリプロトコルである．Checkpoint-Based Rollback-Recovery と異なり，ロールバックリカバリ時に一貫性のあるチェックポイントは必要なく，故障が発生したプロセスだけをロールバックリカバリできる．Log-Based Rollback-Recovery には Pessimistic Logging，Optimistic logging，Causal Logging の 3 種類がある．

3　チェックポイントにおける故障対策

チェックポイントを用いる耐故障方式の場合，チェックポイントを保存した計算機ノードの故障によってチェックポイントが失われ，ロールバックリカバリができなくなる可能性がある．そのため，チェックポイントを用いる耐故障方式では，チェックポイントの安全性を高めることが課題の一つとして上げられる．

本研究ではMPICH-Vclのチェックポイントの管理を行うプロセスであるチェックポイントサーバーを冗長化する手法として，チェックポイントにおける故障の対策を参考にした．

チェックポイントにおける故障の対策にはチェックポイントを複数のノードに保存するMIR（Checkpoint Mirroring），高信頼なノードに全てのチェックポイントを保存するCFS（Central File Server），MIRとCFSを組み合わせた2-level Recovery Scheme，チェックポイントを保存するノードをチェックポインティング毎に変更するSkewed Checkpointingなどの手法が提案されている．以下，それぞれについて詳述する．

3.1　MIR[16]

MIR（Checkpoint Mirroring）はチェックポイントを複数のノードに転送・保存し，チェックポイントを作成したプロセスが実行されているノード自身にもチェックポイントを保存する方式である．

単独故障に対応したMIRの場合，それぞれのノードが保存する他ノードのチェックポイントは1個でよく，チェックポイントオーバーヘッドは小さい．以降，単独故障に対応したMIRを 1-mirror MIRと記述する．

MIRで多重故障に対応するためには，チェックポイントを保存するノード数を増やす．例えばチェックポイントをk個のノードに保存した場合，故障ノードがk個までであればロールバックリカバリを行うことができる．しかし，多重故障に対応するためにチェックポイントを保存するノード数を増やすと，チェックポイントの総量はチェックポイントを保存するノード数に比例して大きくなる．そのため，多重故障に対応したMIRはチェックポイントオーバーヘッドが大きくなってしまう．

3.2　CFS[16]

CFS（Central File Server）は信頼性の高い安全な共有ディスクに全てのチェックポイントを保存する方式である．チェックポインティング時，ロールバックリカバリ

時に全ノードからCFSへ，もしくはCFSから全ノードへのデータ転送が行われるので, CFSへのアクセスが集中し，チェックポインティング・ロールバックリカバリ時間は大きくなる．この時間は，CFSのノード数に大きく影響される．CFSのノード数が多ければチェックポイントを分散して保存することができ，チェックポインティング・ロールバックリカバリ時にCFSにアクセスが集中することを防ぐことができる．しかし，CFSは高価であるので，チェックポインティング・ロールバックリカバリ時のオーバーヘッドを低減するために多くのCFSを設置するとコンピュータクラスタの価格を上げてしまい，コンピュータクラスタの安価であるというメリットを損なってしまう．

3.3 2-level Recovery Scheme[17]

2-level Recovery Schemeは1-mirror MIRとCFSを組み合わせたチェックポインティング方式である．低オーバーヘッドで多重故障に対応する目的で提案された．単独故障のみに対応することを考えた場合，チェックポインティング・リカバリオーバーヘッドが小さい1-mirror MIRを用いることが好ましい．しかし，1-mirror MIRだけでは多重故障に対応できないため，チェックポインティング・リカバリオーバーヘッドは大きいが多重故障に対応可能なCFSを1-mirror MIRの数回おきに行う．そして単独故障時は最新の1-mirror MIRのチェックポイントからロールバックリカバリを行い，多重故障時はCFSのチェックポイントからロールバックリカバリを行う．こうすることで，チェックポイント・ロールバックリカバリオーバーヘッドの大きいCFSの頻度を減らし，発生確率の高い単独故障からは素早く復旧可能になる．

3.4 Skewed Checkpointing[18, 19]

Skewed Checkpointingは1-mirror MIRと同様にチェックポイントを他ノードに転送・保存し，チェックポイントを作成したプロセスが実行されているノード自身にもチェックポイントを保存する方式である．1-mirror MIRと異なり，チェックポイントを保存する他ノードはチェックポインティング毎に変更する．チェックポイントの保存先をチェックポインティング毎に変更することで多重故障に対応可能となる．システムソフトウェアによってチェックポイントの保存先を冗長化し，かつチェックポイント保存先の冗長化によるチェックポイントティングオーバーヘッドの増大を防ぐ目的で考案された．

4 MPICH-Vcl

MPICH-Vcl[11, 14] はMPICH[4] に対してロールバックリカバリプロトコル [9] としてNon-Blocking Coordinated Checkpointingプロトコルを実装したMPIライブラリである．Checkpoint-Based Rollback-Recoveryを実装しているため，ロールバックリカバリ時に一貫性のあるチェックポイントが必要になり，MPIアプリケーションプロセスのチェックポイントが1つでも欠けている場合はロールバックリカバリできない．MPICH-VclはMPICH-Vプロジェクト [15] の一環として作成された．MPICH-Vプロジェクトは，アプリケーションの実行時間に対する影響など，MPIにおけるロールバック・リカバリプロトコルの特性を調査するプロジェクトである．

MPICH-Vclの他に，Pessimistic loggingプロトコルを実装したMPICH-V1[23]，MPICH-V2[22]，Causal loggingプロトコルを実装したMPICH-Vcausal[20]，Blocking Coordinated Checkpointingプロトコルを実装したMPICH-Pcl[12, 13] が実装されている．

4.1 MPICH-Vclの構成

MPICH-Vclのプロセスの構成を図1に示す．MPICH-Vclはディスパッチャー，チェックポイントサーバー，チェックポイントスケジューラー，通信デーモン，MPIプロセスの5種類のプロセスから構成されている．各プロセスの機能を以下に述べる．

ディスパッチャー

MPICH-Vclのプロセスの管理を行う．ディスパッチャーはMPICH-VclでMPIアプリケーションを実行する際，最初に起動されるプロセスである．起動後，他のプロセス（チェックポイントサーバー，チェックポイントスケジューラー，通信デーモン）を起動する．他のプロセスを起動した後は各プロセスのモニターを行う．MPIプロセスの故障を検出した場合，ロールバックリカバリを行う．具体的には，全てのMPIプロセスと通信デーモンを停止させた後，通信デーモンを再起動する．

チェックポイントスケジューラー

後述の通信デーモンに対して定期的にチェックポインティングの指示を出す．チェックポインティングの指示は，デフォルトで5秒に1回となっている．

チェックポイントサーバー

通信デーモンが作成したチェックポイントを保存する．保存したチェックポイントは，通信デーモンが MPI プロセスに故障が発生した際のロールバックリカバリに使用する．デフォルトでは MPI プロセス 5 つにつきチェックポイントサーバーが 1 プロセス実行されるが，オプションで変更することができる．チェックポイントサーバーは複数実行することとなるが，これは 1 つのチェックポイントサーバーにアクセスが集中することを防ぐ負荷分散のためのものであり，チェックポイントサーバーに耐故障性を負荷するための機能ではない．

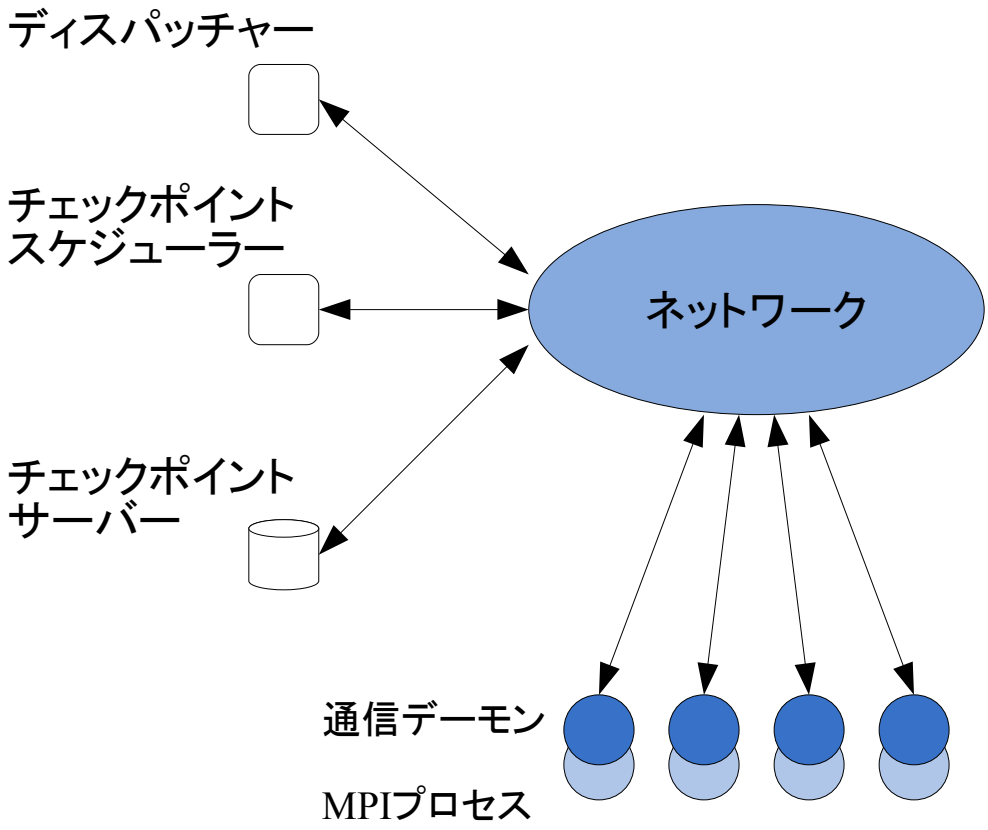


図 1: MPICH-Vcl のプロセスの構成

通信デーモン

MPI プロセスと他の通信デーモン，チェックポイントサーバー，チェックポイントスケジューラー，ディスパッチャー間の通信を担当する．MPICH-Vcl が実行されるとディスパッチャーに起動された後，MPI プロセスを起動する．チェックポイントスケジューラーからチェックポイントの指示を受けた場合は，チェックポイントを作成する．作成したチェックポイントはチェックポイントサーバーに送信・保存する．MPI プロセスのモニターを行い，MPI プロセスに故障が発生した場合，故障の発生をディスパッチャーに伝える．故障が発生した場合，ディスパッチャーに強制終了された後，再起動され，チェックポイントサーバーに保存されたチェックポイントを用いて MPI プロセスをチェックポイントからロールバックリカバリする．

MPI プロセス

MPI アプリケーションのプロセスである．MPI プロセス間の通信は通信デーモンを通じて行われる．

4.2 MPICH-Vcl の動作

MPICH-Vcl の起動から実行終了までの処理，チェックポインティング，ロールバックリカバリの処理について説明する．

MPICH-Vcl の起動・終了

MPICH-Vcl の起動から終了までの処理を図 2 に示す．

1. mpirun コマンドが入力される．
2. シェルスクリプトからディスパッチャーが起動される．
3. コマンドラインからマシンファイルなどの情報を受取る．
4. 変数の初期化を行う．
5. チェックポイントサーバー，チェックポイントスケジューラーを起動する．
6. チェックポイントサーバー，チェックポイントスケジューラーをモニターするスレッド `server_monitor` を作成する．

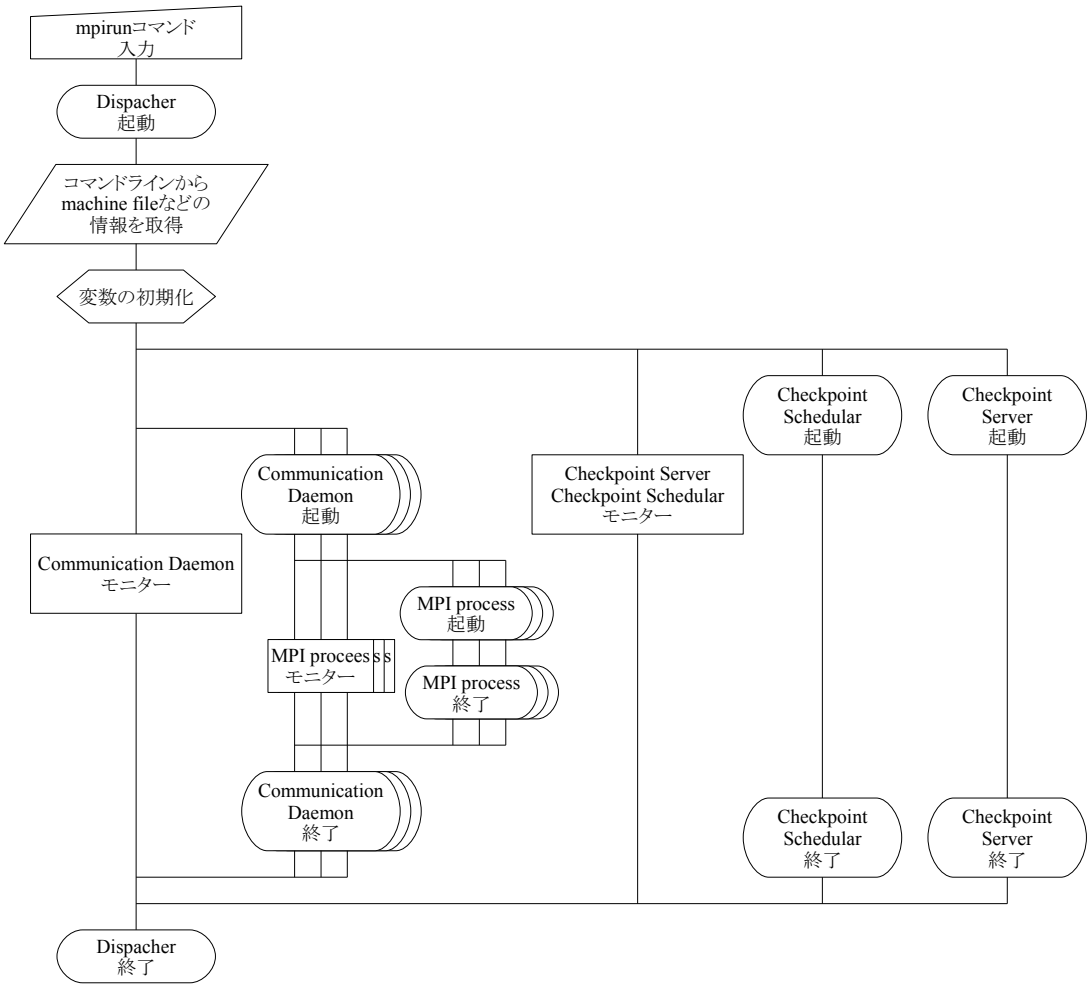


図 2: MPICH-Vcl の起動から終了までの処理の流れ

7. 通信デーモンを起動する．起動された通信デーモンはMPI プロセスを起動する．
8. 各プロセスのモニターを行い，MPI プロセスの終了を待つ．
9. チェックポイントサーバー，チェックポイントスケジューラーを停止する．
10. ディスパッチャーを停止する．

チェックポインティング

チェックポインティングの動作を図 3 に示す．

1. チェックポイントスケジューラーがチェックポイントタグを送信する．

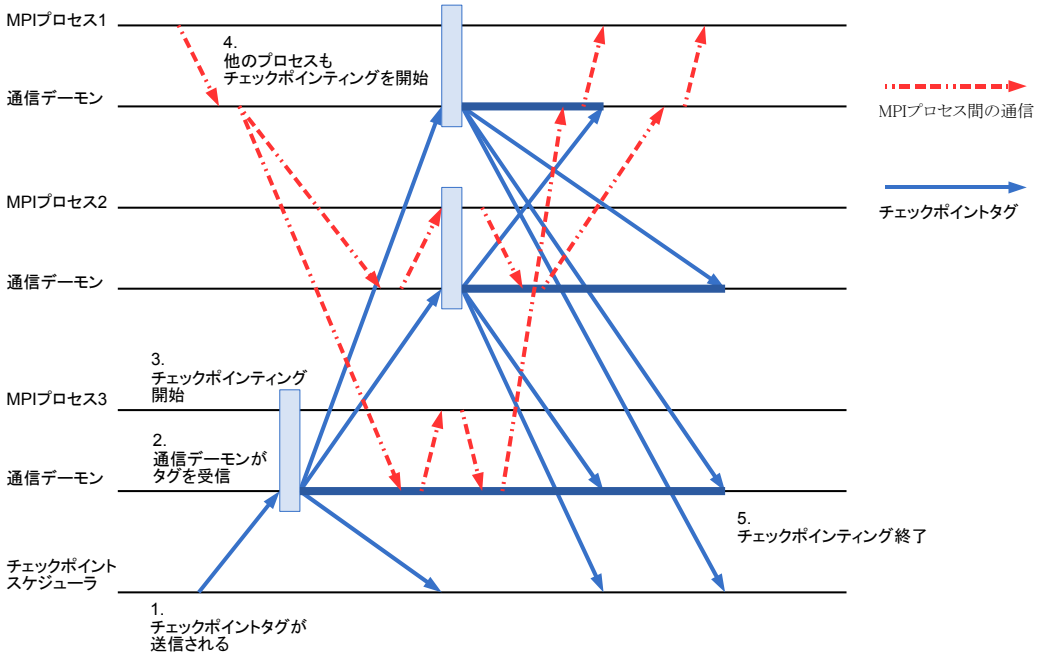


図 3: MPICH-Vcl のチェックポインティングの動作

2. チェックポイントタグを通信デーモンが受信する．

3. チェックポイントタグを受信した通信デーモンはチェックポイントの作成を開始する．同時に，他の全ての通信デーモンに対してチェックポイントタグを送信する．チェックポインティング時も MPI プロセスは実行される．チェックポインティング時の MPI プロセス間の通信はチェックポイントの一部として保存される．

4. 他の通信デーモンからのチェックポイントタグを受信した通信デーモンも 3. と同じ処理を行う．

5. 全ての通信デーモンからチェックポイントタグを受信すると，チェックポインティングを終了する．

6. 作成したチェックポイントはチェックポイントサーバーに転送する．

ロールバックリカバリ

MPICH-Vcl のロールバックリカバリの動作を図 4 に示す．

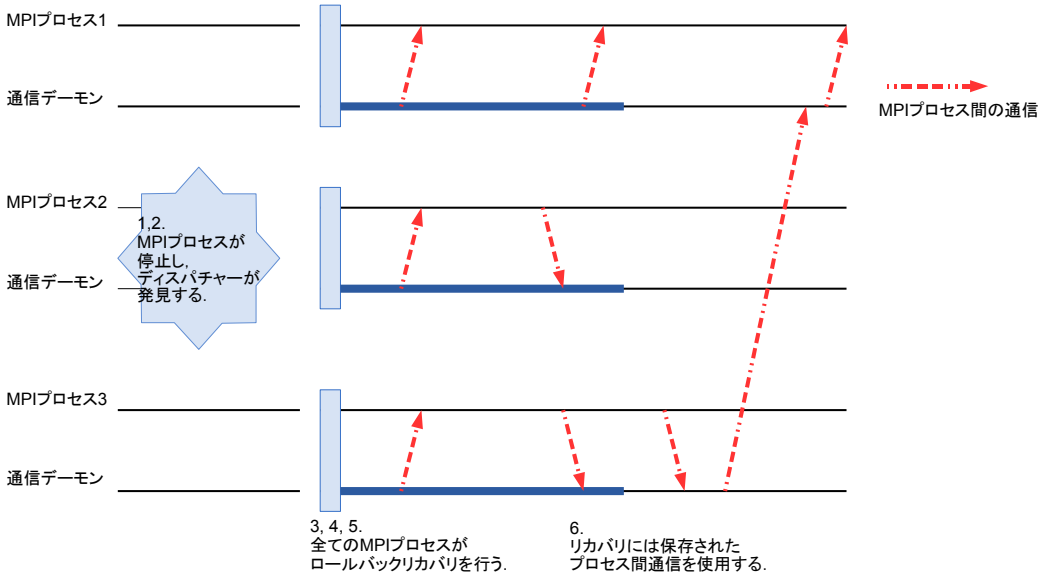


図 4: MPICH-Vcl のロールバックリカバリの動作

1. 何らかの原因で MPI プロセスに故障が発生する．
2. ディスパッチャーが故障を検出する．
3. ディスパッチャーは全ての通信デーモンを強制終了する．通信デーモンは MPI プロセスを強制終了する．
4. ディスパッチャーが通信デーモンを再起動する．
5. 再起動された通信デーモンは，チェックポイントサーバーに保存したチェックポイントの中で最も新しいチェックポイントを使用して MPI プロセスをロールバックリカバリする．
6. ロールバックリカバリされた MPI プロセスは最初，チェックポイントに保存された MPI プロセス間の通信を使用して実行し，その後通常の実行に戻る．

5 MPICH-Vclの特性と問題点

MPICH-Vcl におけるチェックポインティングにかかるオーバーヘッドと，MPI プロセスに対して耐故障性を付加するために導入されたプロセスに故障が発生した場合に，MPI アプリケーションの実行に与える影響について調査した．また，調査の結果から MPICH-Vcl の問題点を検討した．

以下に調査方法と結果，MPICH-Vcl の問題点についてまとめる．

5.1 チェックポインティングによるオーバーヘッド

MPICH-Vcl におけるチェックポインティングオーバーヘッドを調査した結果について述べる．MPICH-P4 と MPICH-Vcl の 2 種類の MPI ライブラリで NAS Parallel Benchmark[26] を実行し，実行時間を比較した．調査の結果，MPICH-Vcl におけるロールバックリカバリプロトコルによる MPI アプリケーションの実行時間に対するオーバーヘッドは，10%以下であることを確認した．

次に調査環境を表 1 に示す．調査に使用した MPI ライブラリは MPICH-Vcl と MPICH-P4 である．PC クラスタのノード数は 8 ノードで，1 ノードにつき 2 つのプロセッサを搭載している．

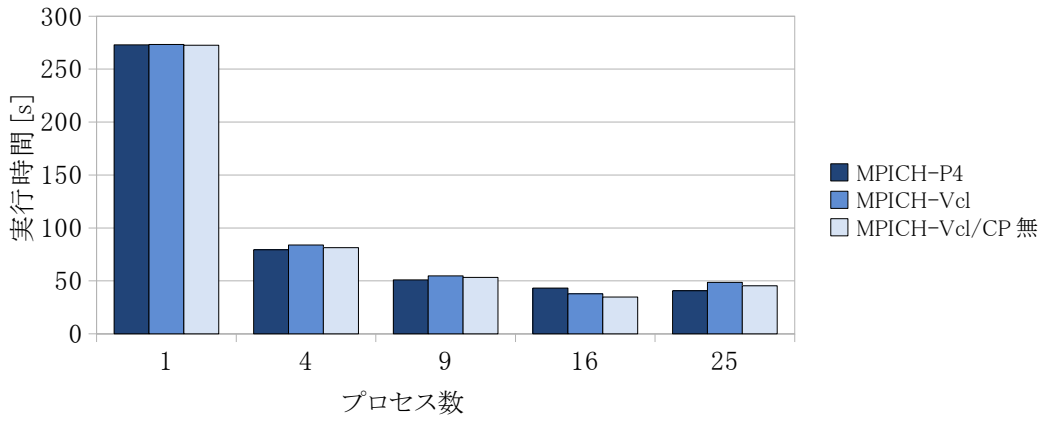
ベンチマークには NAS Parallel Benchmark LU，BT，SP，Class=A を使用した．

表 1: MPICH-Vcl におけるチェックポインティングオーバーヘッドの調査環境

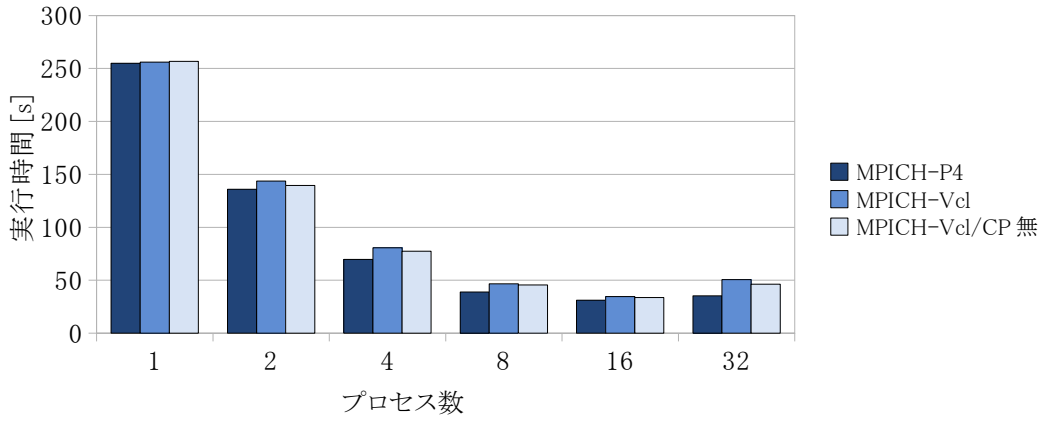
CPU	Intel(R) Xeon(TM) CPU 2.80GHz (2/node)
Memory	1GByte
OS	CentOS release 4.7
Kernel	Linux 2.6.9
MPICH	mpich-1.2.5.2
ノード数	8

測定の結果は次の通りである．

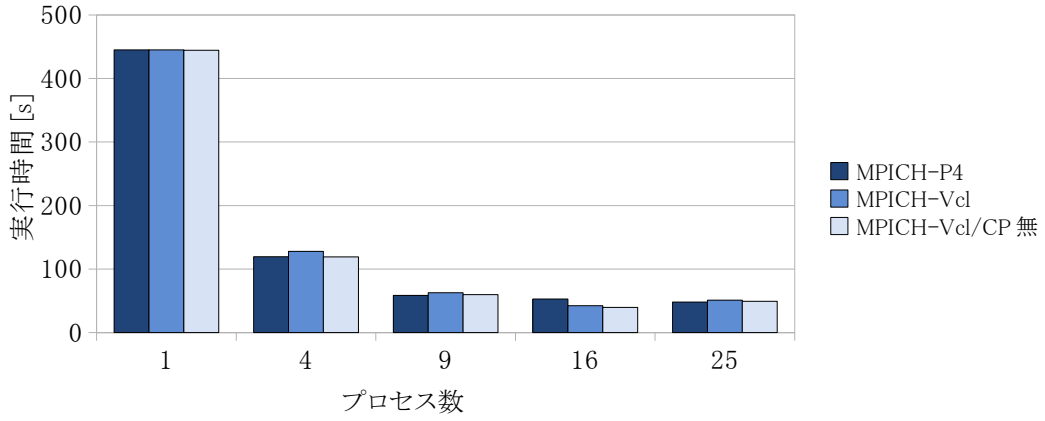
MPICH-P4，MPICH-Vcl，MPICH-Vcl（チェックポインティング無し）における NAS Parallel Benchmark LU，BT，SP の実行時間の測定結果を図 5 に示す．図 5 からは，MPICH-Vcl と MPICH-P4 の間に大きな性能差は確認できない



(a) NAS Parallel Benchmark BT Class=A



(b) NAS Parallel Benchmark LU Class=A



(c) NAS Parallel Benchmark SP Class=A

図 5: MPICH-P4，MPICH-Vcl，MPICH-Vcl（チェックポインティング無）における NAS Parallel Benchmark BT，LU，SP 実行時間の比較

次に，NAS Parallel Benchmark の実行時間測定結果から算出した MPICH-P4, MPICH-Vcl，MPICH-Vcl（チェックポインティング無し）における NAS Parallel Benchmark の実行時間の比較を，表 3，表 2，表 4 に示す．

各表における Vcl/P4 は MPICH-Vcl と MPICH-P4 の MPI アプリケーション実行時間の比較である．

同様に Vcl（CP 無）/P4 は MPICH-Vcl がチェックポインティングを行わない場合の MPI アプリケーションの実行時間と，MPICH-P4 による MPI アプリケーション実行時間の比較である．これは MPICH-P4 と MPICH-Vcl のアーキテクチャの違いによる性能差を表す．

Vcl/Vcl（CP 無）は MPICH-Vcl がチェックポインティングを行わずに MPI アプリケーションを実行した場合に対し，チェックポインティングを行うことで増加した実行時間の割合である．これは MPICH-Vcl がチェックポインティングにかかるオーバーヘッドを表す．

表 3，表 2，表 4 より，MPICH-Vcl と MPICH-P4 の性能差はアプリケーションによって異なり，MPICH-Vcl が MPICH-P4 よりも性能がよい場合もあることが確認できる．これは，MPICH-Vcl と MPICH-P4 のアーキテクチャの違いによるものと考えられる．また，実験に使用した MPI アプリケーションに限れば，MPICH-Vcl がチェックポインティングにかかるオーバーヘッドは 10[%] 以下である．

表 2: MPICH-P4，MPICH-Vcl，MPICH-Vcl（CP 無）の NAS Parallel Benchmark 実行時間の比較（BT）

プロセス数	P4:Vcl[%]	P4:Vcl（CP 無）[%]	Vcl:Vcl（CP 無）[%]
1	0.03	-0.11	0.14
4	7.11	-0.19	7.31
9	7.23	1.77	5.36
16	-19.91	-24.79	6.49
25	6.24	2.14	4.01

表 3: MPICH-P4，MPICH-Vcl，MPICH-Vcl（CP 無）の NAS Parallel Benchmark 実行時間の比較（LU）

プロセス数	P4:Vcl[%]	P4:Vcl（CP 無）[%]	Vcl:Vcl（CP 無）[%]
1	0.41	0.67	-0.26
2	5.74	2.75	2.91
4	15.63	11.08	4.10
8	20.25	17.40	2.43
16	10.85	7.99	2.64
32	43.16	31.02	9.26

表 4: MPICH-P4，MPICH-Vcl，MPICH-Vcl（CP 無）の NAS Parallel Benchmark 実行時間の比較（SP）

プロセス数	P4:Vcl[%]	P4:Vcl（CP 無）[%]	Vcl:Vcl（CP 無）[%]
1	0.11	-0.13	0.24
4	5.65	2.43	3.15
9	7.49	4.70	2.67
16	-12.35	-19.50	8.89
25	19.88	11.59	7.43

5.2 耐故障性を向上のためのプロセスにおける故障

MPICH-Vcl のプロセス中で耐故障性に関わっている管理プロセスに故障が発生した場合, MPI アプリケーションの実行に与える影響を調査した. 調査の結果, MPICH-Vcl は MPI アプリケーション以外のプロセスの故障に対応していないことを確認した.

調査環境を表 5 に示す. MPICH-Vcl で MPI アプリケーションを実行中にディスパッチャー, チェックポイントスケジューラー, チェックポイントサーバー, 通信デーモンの各プロセスを kill コマンドで停止させ, MPI アプリケーションの実行に与える影響を調査した. 実行する MPI アプリケーションには NAS Parallel Benchmark LU Class=A プロセス数=2 を使用した.

表 5: 調査環境

CPU	Intel(R) Pentium(R) III 1400MHz (2/node)
Memory	1GByte
OS	Red Hat Linux release 9
Kernel	2.4.20-8smp
MPICH	mpich-1.2.5.2
ノード数	2

各プロセスに故障が発生した場合の調査結果を以下に示す.

ディスパッチャーの故障 ディスパッチャーに故障が発生した場合, MPI アプリケーションの実行は継続される. しかし, MPI プロセスに故障が発生してもロールックリカバリができなくなる. また, ディスパッチャーは再実行されない.

チェックポイントスケジューラーの故障 チェックポイントスケジューラーに故障が発生しても MPI アプリケーションは実行し続けることができる. ただ, チェックポイントスケジューラーが停止すると新しいチェックポインティングは行われないため, MPI プロセスのロールバックリカバリを行う場合はチェックポイントスケジューラーが停止する以前のチェックポイントのなかで最も新しいチェックポイントが毎回使われる. チェックポイントスケジューラーに故障が発生しても, ディスパッチャーはチェックポイントスケジューラーの再実行は行わない.

チェックポイントサーバーの故障 チェックポイントサーバーに故障が発生した場合, 故障が発生するタイミングによって影響が異なる. チェックポイントサーバー

に故障が発生するタイミングにはチェックポインティングを行っている最中とそうではない場合の 2 通りがある．

チェックポインティングの最中の故障 チェックポインティング中にチェックポイントサーバーに故障が発生した場合，通信デーモンとチェックポイントサーバーとの接続がきれ，チェックポインティングの処理が中断されてしまい，MPI アプリケーションの実行を続けることはできなくなる．

チェックポインティングの最中ではない場合の故障 チェックポインティングを行っていない状態でチェックポイントサーバーに故障が発生した場合，MPI アプリケーションの実行を続けることができる．しかし，チェックポイントサーバーに故障が発生した後でチェックポイントスケジューラーがチェックポインティングの指示を出した場合，通信デーモンとチェックポイントサーバーとの接続ができず，MPI アプリケーションの実行を続けることはできなくなる．チェックポイントスケジューラーに故障が発生した後であれば，チェックポイントサーバーに故障が発生しても MPI アプリケーションの実行を続けることは可能である．

チェックポイントサーバーに故障が発生しても，ディスパッチャーはチェックポイントサーバーを再実行しない．

通信デーモンの故障 Communicaton Daemon に故障が発生した場合，MPI プロセスの故障として扱われ，ロールバックリカバリが行われる．

以上のように，MPICH-Vcl は一部の管理プロセスにおける故障に対応できない．一部の管理プロセスに故障が発生した場合，MPI プロセスの耐故障性が失われ，MPI アプリケーションの実行が停止した．このことは MPICH-Vcl の問題点である．

5.3 MPICH-Vcl の問題点

MPICH-Vcl の問題点についてまとめると，MPICH-Vcl の問題点は MPI プロセスに耐故障性を付加するために導入された管理プロセスにおける故障に対応していないことである．

調査の結果，MPICH-Vcl の MPI アプリケーション実行時間をチェックポインティングを行う場合と行わない場合で比較して得た，MPICH-Vcl のチェックポインティングによるオーバーヘッドは 10[%] 以下である．チェックポインティングによるオーバーヘッドにのみ注目した場合，10[%] 程度のオーバーヘッドで MPI アプリケーショ

ンに対して耐故障性を付加できるのであれば，耐故障 MPI ライブラリとして有用であると判断した．

一方で MPI アプリケーションの耐故障性を向上させるために導入された管理プロセスには冗長性は無く，MPI アプリケーションを実行中に管理プロセスが停止した場合，MPI プロセスの耐故障性が失われ，MPI アプリケーションの実行が停止してしまう．従って，MPI プロセスに対して耐故障性を付加するために導入されたプロセスについても耐故障性が必要であると考える．

6　チェックポイントサーバー冗長化の方針

本研究では，MPICH-Vcl の耐故障性の向上として，まずチェックポイントサーバーにおける故障の対策を行う．

6.1　チェックポイントサーバーにおける故障への対策

チェックポイントサーバーにおける故障への対策としては，チェックポイントサーバーの冗長化が良いと考えた．

冗長化以外のチェックポイントサーバーにおける故障への対策としては，チェックポイントサーバーの再実行があげられる．しかし，チェックポイントサーバーを再実行するだけでは MPI プロセスとチェックポイントサーバーに同時に故障が発生した場合に対応できない．チェックポイントサーバーはチェックポイントを管理するプロセスであるので，チェックポイントサーバーに故障が発生するとチェックポイントサーバーが管理していたチェックポイントも失われ，MPI プロセスをロールバックリカバリできない．従って，チェックポイントサーバーの故障対策として，チェックポイントサーバーの故障発生時にチェックポイントサーバーを再実行させるだけでは，MPI プロセスとチェックポイントサーバーの同時故障に対応できないため，故障対策としては不十分である．

そのため，MPI プロセスとチェックポイントサーバーに同時に故障が発生した場合に対応するには，チェックポイントサーバーに故障が発生した場合にもチェックポイントの安全性を確保する必要がある．チェックポイントサーバーを複数用意（冗長化）し，MIR や Skewed Checkpointing のようにチェックポイントを冗長化することで，チェックポイントサーバーに故障が発生してもチェックポイントの安全性を確保できる．

以上の理由により，チェックポイントの安全性を確保することができ，なおかつチェックポイントサーバーの故障に対応できる手法は冗長化であると判断した．

6.2　チェックポイントサーバー冗長化の方式

チェックポイントの管理方式は，**単純冗長化**，**MIR**，**CFS**，**2level Recovery Scheme**，**Skewed Checkpointing**，**チェックポイントサーバー切替方式**の 6 通りを考えた．それぞれの方式を用いてチェックポイントの安全性を確保した場合の効果について述べる．

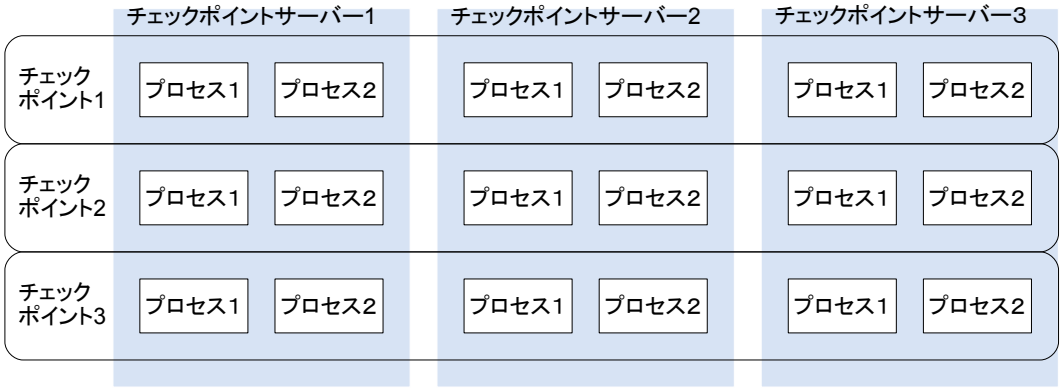


図 6: チェックポイントサーバーにおける単純な冗長化

単純冗長化 この方式では複数のチェックポイントサーバーを実行し，全てのチェックポイントサーバーに同じチェックポイントを保存する．

図 6 は 2 プロセスの MPI アプリケーションにおいて，3 つのチェックポイントサーバーを実行し，3 回のチェックポインティングを行った場合に，各チェックポイントサーバーに保存されるチェックポイントを示す．

チェックポイントはチェックポイントサーバーの数だけ冗長化できる．そのため，故障が発生するチェックポイントサーバーに関係なく必ず最新のチェックポイントを利用できる．一方で，チェックポイントを複数の計算機ノードに転送するため，チェックポインティングのオーバーヘッドが大きくなる．

MIR チェックポイントの安全性を確保する手法として MIR を用いた場合，1-mirror MIR ではチェックポインティングにかかるオーバーヘッドは小さく抑えつつチェックポイントサーバーを冗長化できる．ただし，複数のチェックポイントサーバーで故障が発生したとき，Coordinated Checkpointing でロールバックリカバリを行うときに必要なチェックポイントの一貫性（全てのプロセスが同じタイミングでとったチェックポイント）が保てなくなり，ロールバックリカバリを行うことができない場合がある．複数のチェックポイントサーバーにおける故障に対応するためには，チェックポイントを保存するチェックポイントサーバーを増やさなければならないので，チェックポイントオーバーヘッドが増加する．

CFS チェックポイントの安全性を確保する手法として CFS を用いた場合，現在の MPICH-Vcl の実装を変更せずにチェックポイントの安全性を確保できる．ただ

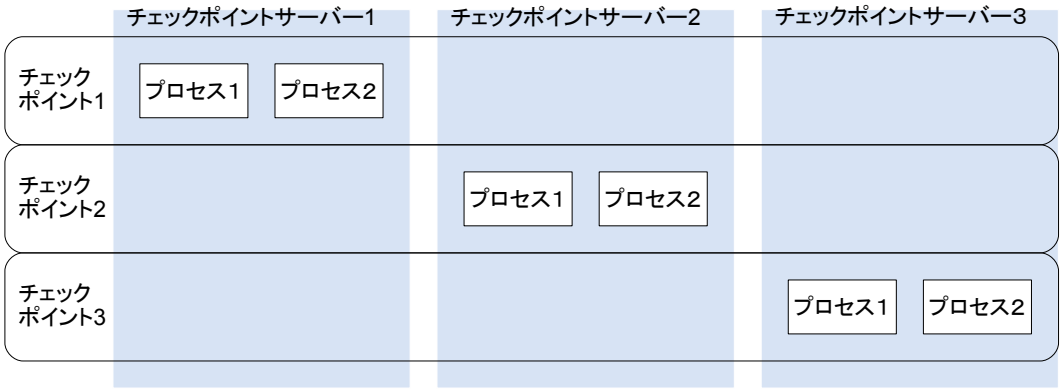


図 7: チェックポイントサーバー切替方式

し，信頼性の高い計算機ノードを導入するためのコストがかかり，コンピュータクラスタの安価であるというメリットが損なわれる．

2-level Recovery Scheme チェックポイントの安全性を確保する手法として 2-level Recovery Scheme[17] を用いた場合，CFS を用いた場合と同様に，信頼性の高い計算機ノードを導入するためのコストがかかり，コンピュータクラスタの安価であるというメリットが損なわれる．CFS よりチェックポインティングにかかるオーバーヘッドは小さい．

Skewed Checkpointing チェックポイントの安全性を確保する手法として Skewed Checkpointing を用いた場合，チェックポインティングにかかるオーバーヘッドは 1-mirror MIR と同程度になる．複数の計算機ノードに故障が発生した場合，1-mirror MIR よりチェックポイントの一貫性が失われにくいので，1-mirror MIR より耐故障性が高い．

チェックポイントサーバー切替方式 この方式はチェックポイントサーバーを複数用意し，チェックポインティングごとにチェックポイントを保存する計算機ノードを切替える．

図 7 は 2 プロセスの MPI アプリケーションを実行し，3 回のチェックポイントを行った場合に各チェックポイントサーバーが保存しているチェックポイントを表している．

単純冗長化と異なるのは複数のチェックポイントサーバーに同じチェックポイントを保存しないことである．そのため単純な冗長化ほどチェックポイントの

オーバーヘッドがかからない点にある．

Skewed Checkpointing と異なる点は，あるタイミングでとったチェックポイントは 1 つの同じ計算機ノードに保存されることである．チェックポインティングにかかるオーバーヘッドは，チェックポイントサーバーを単純に冗長化し，複数の計算機ノードに同じタイミングで取ったチェックポイントを保存する方式よりは小さくなるが，Skewed Checkpointing や 1-mirror MIR よりは大きくなると予想される．

MPICH-Vcl に実装することを考えた場合，チェックポイントサーバー切替方式は Skewed Checkpointing と比べ単純な実装にすることができる．また，複数の計算機ノードに故障が発生した場合に Skewed Checkpointing よりもチェックポイントの一貫性が失われにくく，Skewed Checkpointing よりも耐故障性は高い．

本研究では，まず MPICH-Vcl に対してチェックポイントサーバー切替方式を実装することとした．

7　チェックポイントサーバー切替方式の実装

本章ではチェックポイントサーバー冗長化のために MPICH-Vcl に対して行った具体的な実装について述べる．

7.1　プロセスの構成と役割

チェックポイントサーバーを冗長化した際のプロセスの構成を図 8 に示す．図 8 に示すようにチェックポイントサーバーを複数実行し，冗長化している．

図 8 の構成で MPI アプリケーションを実行した場合に，各プロセスが計算機ノードに割り当てられた様子を図 9 に示す．

ディスパッチャー，チェックポイントサーバー，通信デーモンにチェックポイントサーバーの切替をするための機能を追加した．ディスパッチャー，チェックポイン

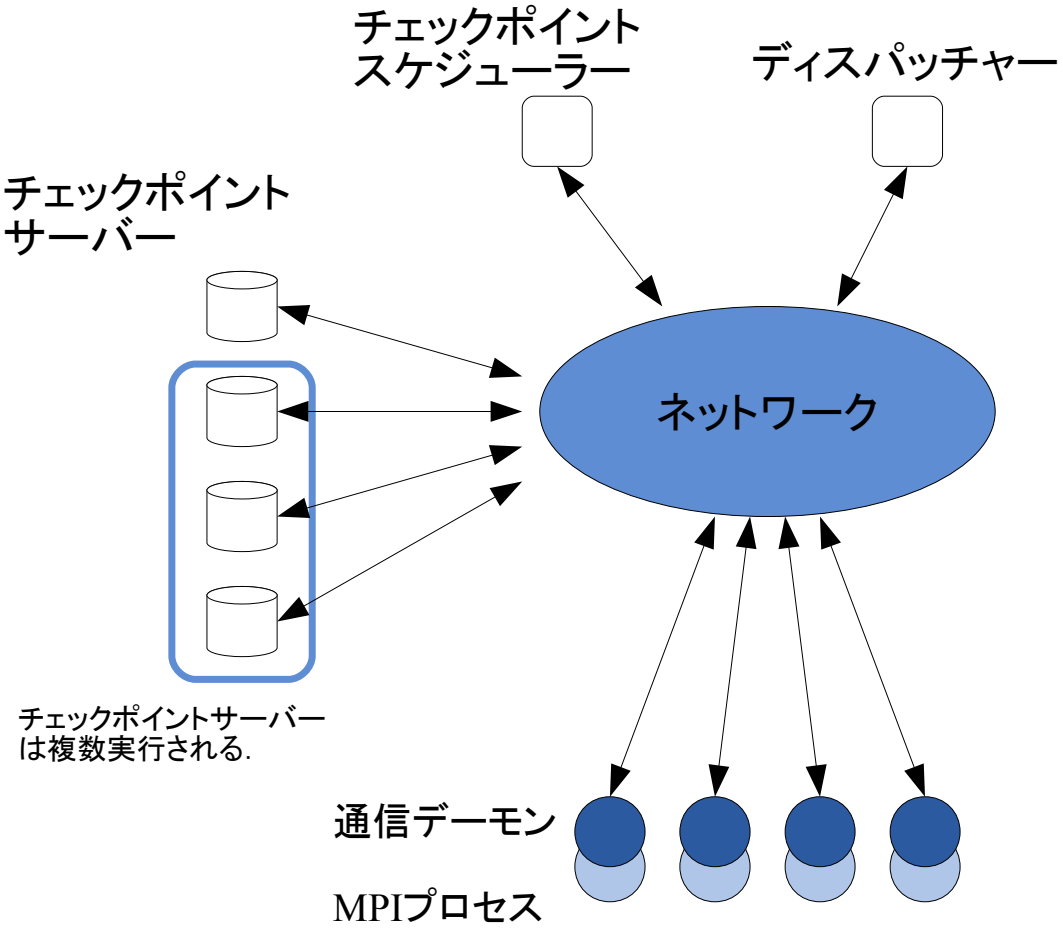


図 8: MPICH-Vcl に対してチェックポイントサーバー冗長化を行ったプロセスの構成

トサーバー，通信デーモンの各プロセスの機能の変更点を以下に述べる．

ディスパッチャー

ユーザーが指定した数だけチェックポイントサーバーを実行する．MPICH-Vcl は MPI プロセス 5 つに対して 1 つのチェックポイントサーバーを実行していた．チェックポイントサーバー冗長化の実装では MPI プロセスの数に関係なくチェックポイントサーバーを複数実行する．

チェックポイントサーバー

MPICH-Vcl は MPI プロセス 5 つに対して 1 つのチェックポイントサーバーが実行されていた．チェックポイントサーバー冗長化の実装では MPI プロセスの数に関係なく複数実行されるように変更する．

通信デーモン

通信デーモンには，チェックポインティング時にチェックポイントを保存するチェックポイントサーバーや，ロールバックリカバリ時に使用するチェックポイントを決めるといったチェックポイントの管理を行う機能を追加する．これらの通信デーモンに対して新しく追加した機能によって，チェックポインティング時にチェックポイ

図 9: Checkpoitn Server を冗長化した際の計算機ノードに対するプロセスの割り当て

ントサーバーを切替え，ロールバックリカバリ時に最新のチェックポイントを選択する．通信デーモンは実行されている全てのチェックポイントサーバーの IP アドレスとポート番号を保持する．

7.2 チェックポインティングの動作と実装

故障が発生していない場合のチェックポイントサーバー切替方式の動作を説明する．

図 10 に 3 つのチェックポイントサーバーを使用して実行した場合の，チェックポインティングの動作を示す．黒い枠は，各チェックポインティング時にチェックポイントが保存されるチェックポイントサーバーを表す．

1. 1 回目のチェックポインティングではチェックポイントサーバー 1 にチェックポイントを保存する．
2. 2 回目のチェックポインティングではチェックポイントサーバー 2 にチェックポイントを保存する．
3. 同様に，3 回目以降もチェックポイントを保存するチェックポイントサーバーを切り替える．
4. 全てのチェックポイントサーバーにチェックポイントを保存したら 1. に戻る．

故障が発生した場合，すなわちチェックポインティング時に，チェックポイントを保存しようとしたチェックポイントサーバーが停止していた場合の動作を以下に説明する．

図 11 は MPI アプリケーション実行中にチェックポイントサーバーが故障した場合の例である．3 つのチェックポイントサーバーを使用して MPI アプリケーションを実行している最中に，チェックポインティング 3 のタイミングでチェックポイントサーバー 2 が停止した場合である．

1. 3 回目のチェックポインティングとほぼ同じタイミングでチェックポイントサーバー 2 が停止．
2. チェックポインティング 4 はチェックポイントサーバー 2 が停止したことによる影響を受けない．
3. チェックポインティング 5 ではチェックポイントサーバー 2 が使用できないため，チェックポイントサーバー 3 にチェックポインティング 5 で作成したチェックポイントを保存する．

4. 以降は，チェックポイントサーバー 1 とチェックポイントサーバー 3 を交互に使用してチェックポインティングを行う．

次に，チェックポインティング毎に Checkpiont Server を切替えるために，MPICH-Vcl に対して行ったチェックポイントサーバー切替方式の実装を説明する．

チェックポイントを保存する Checkpiont Server は通信デーモンが自律的に決める．通信デーモンの変更点を以下に述べる．

通信デーモンにおけるチェックポインティングの処理を以下に示す．

1. チェックポインティングが開始されるとチェックポイントサーバーに接続する．

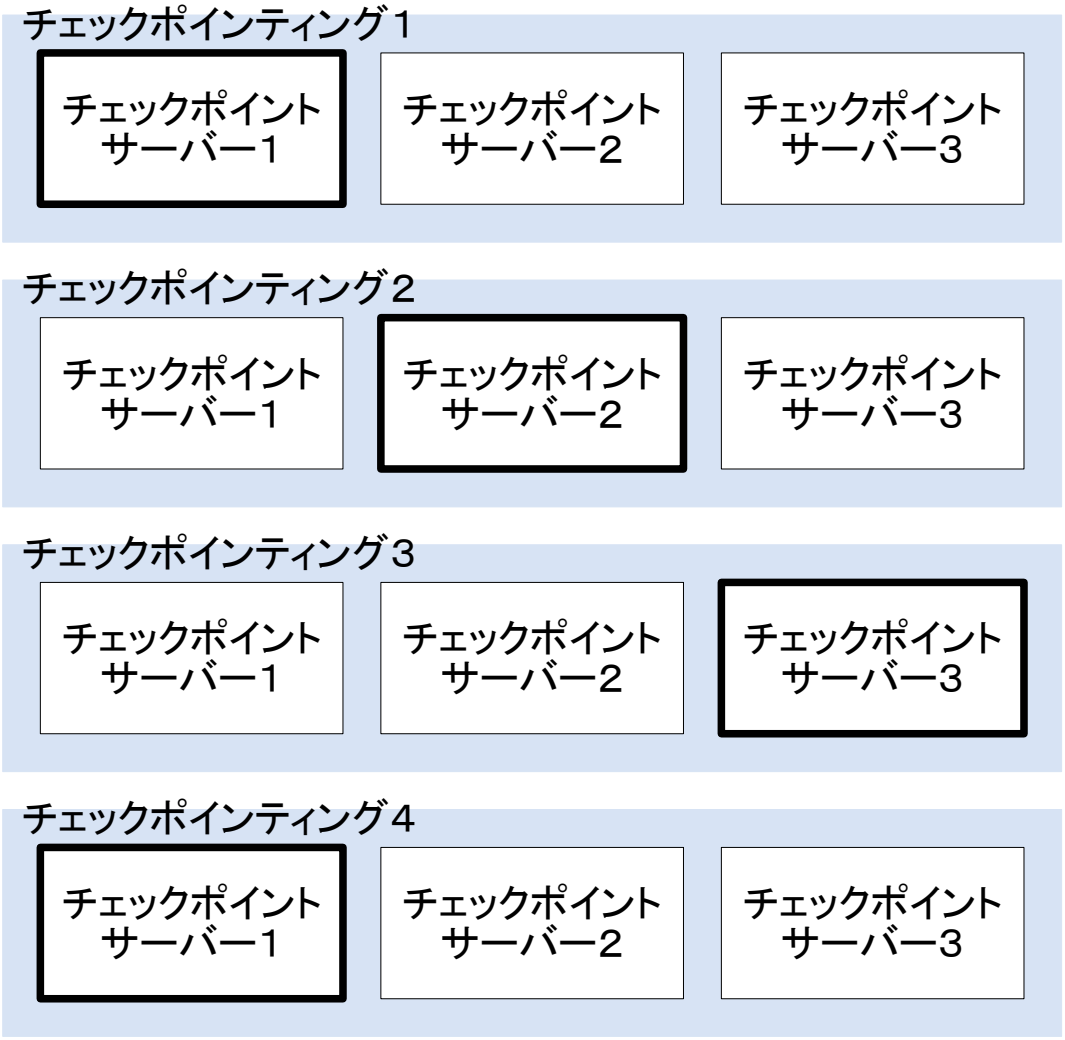


図 10: チェックポインティングの動作

2. 通信デーモンとチェックポイントサーバーの接続はチェックポインティングが終了するまで維持される.

3. チェックポインティングが終了すると通信デーモンはチェックポイントサーバーとの接続を断つ.

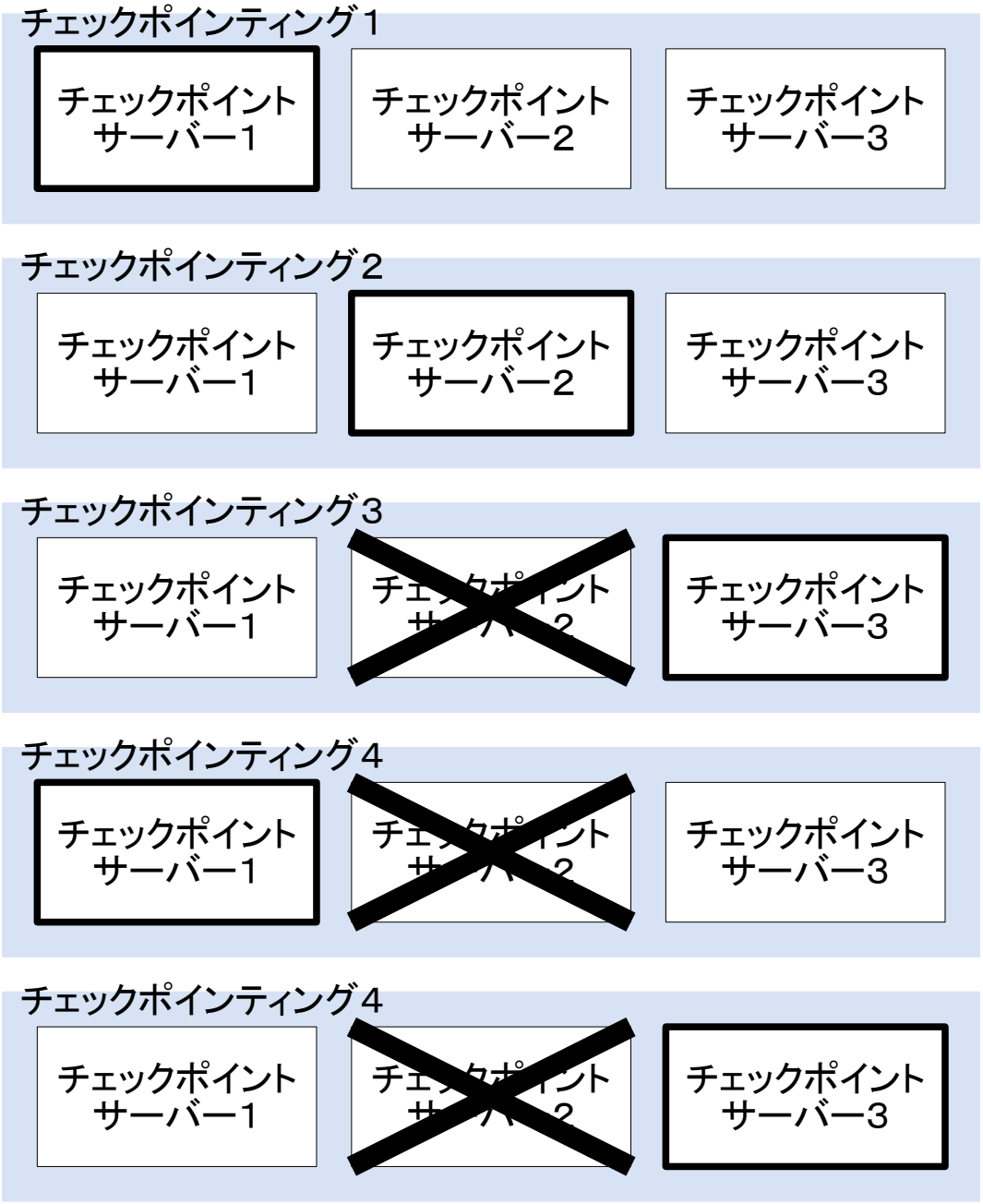


図 11: チェックポイントサーバーが停止した場合の動作

チェックポインティングが開始され，通信デーモンがチェックポイントサーバーと接続する部分を，図 12 のフローチャートに示した処理に変更する．

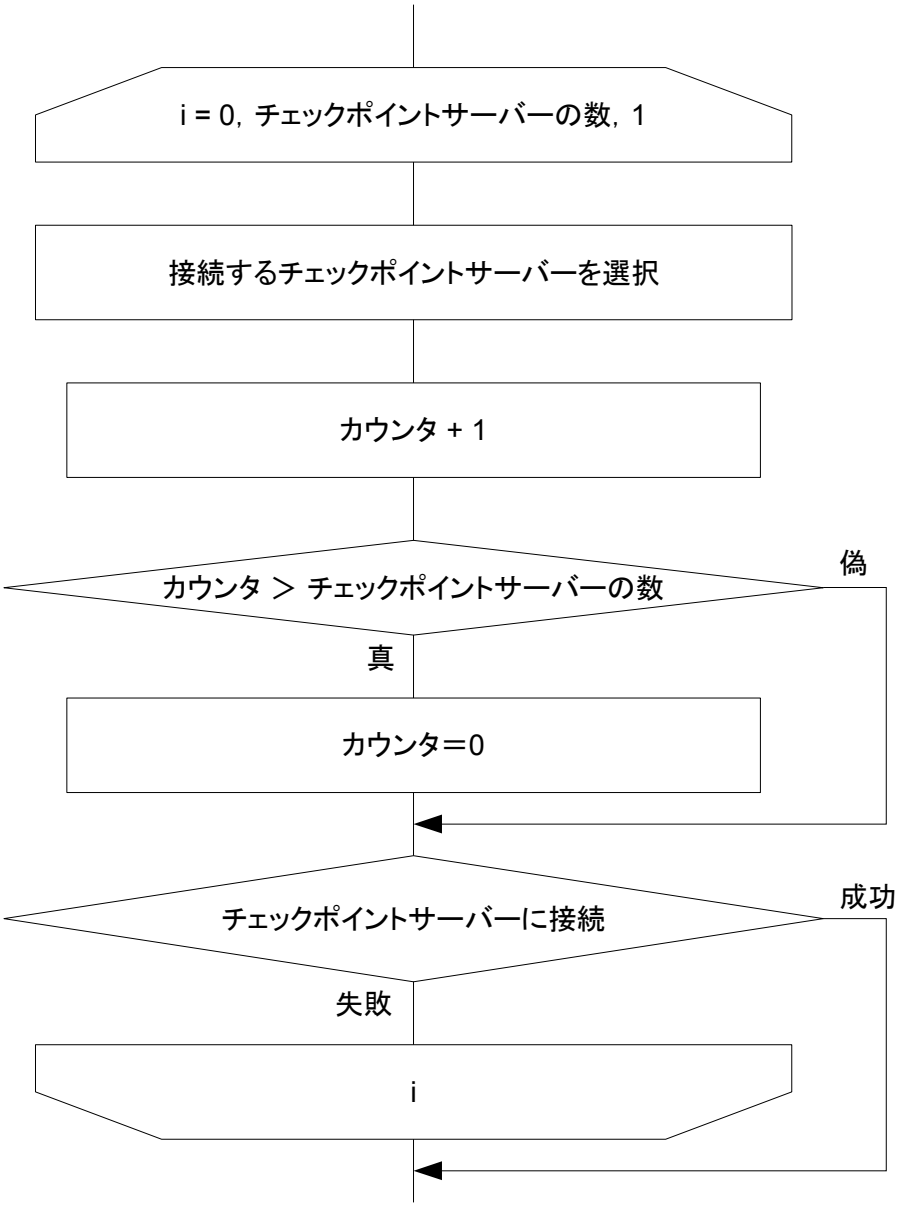


図 12: チェックポイントサーバーを切替える処理

7.3　ロールバックリカバリの動作と実装

MPI プロセスに故障が発生した場合のロールバックリカバリの動作を説明する．

図 13 にチェックポイントサーバーを冗長化した際のロールバックリカバリの動作を示す．図 13 は 4 つのチェックポイントサーバーを実行し，6 回目のチェックポインティングを行った後に MPI プロセスの故障が発生した状態を示している．MPI プロセスのロールバックリカバリには利用可能なチェックポイントの内，最新のチェックポイントを使用する．図 13 の場合，チェックポイント 6 を使用し，MPI プロセスのロールバックリカバリを行う．

故障がない状態での最新のチェックポイントを探す手順を以下に示す．

1. まず，チェックポイントサーバー 4 にチェックポイント 6 が存在を確認する．
2. チェックポイントサーバー 4 にチェックポイント 6 が存在しなかった場合，チェックポイントサーバー 3 にチェックポイント 6 が存在を確認する．チェックポイントサーバー 3 にもチェックポイントが存在しなかった場合，さらに他のチェックポイントサーバーにチェックポイント 6 の存在の確認を行う．
3. チェックポイント 6 を発見した場合，チェックポイント 6 を使用して MPI プロセスをロールバックリカバリする．

チェックポイントサーバーに故障が発生した場合，すなわち MPI プロセスのロールバックリカバリ時のチェックポイントサーバー停止により，最新のチェックポイントを発見できない場合を説明する．

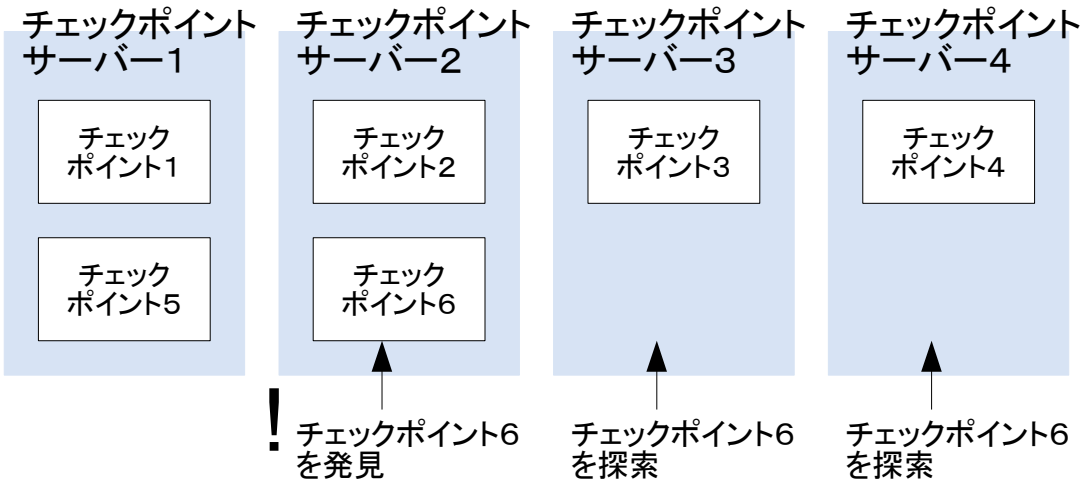
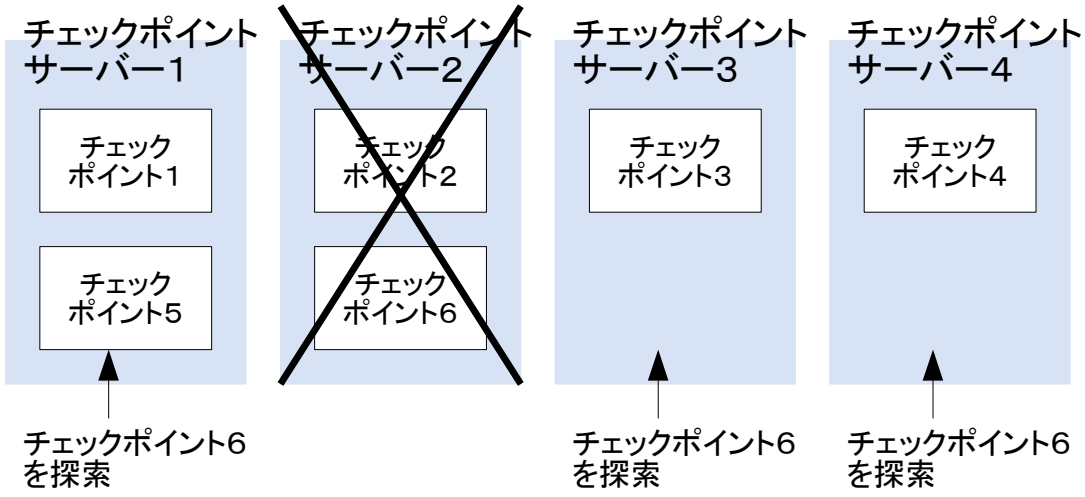
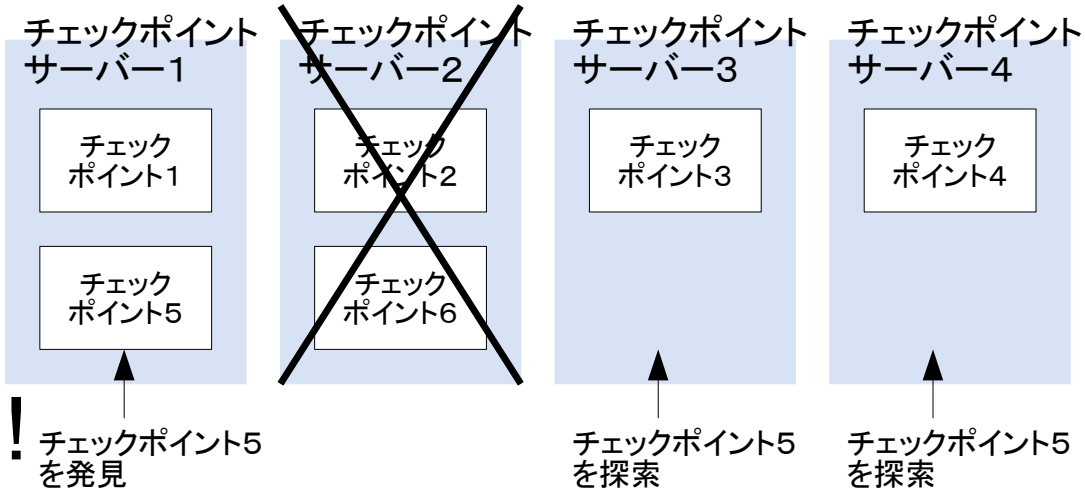


図 13: ロールバックリカバリの動作

図 14(a) は 4 つのチェックポイントサーバーを実行して冗長化を行った場合のロールバックリカバリの例である．6 回目のチェックポインティングの後に MPI プロセスに故障が発生し，MPI プロセスのロールバックリカバリを行おうとした際，最新のチェックポイントであるチェックポイント 6 を保存していたチェックポイントサーバーが停止していたために，チェックポイント 6 を発見できなかった状態を表して



(a) 最新のチェックポイントが発見できない場合



(b) 1 つ前のチェックポイントにさかのぼる場合

図 14: ロールバックリカバリの動作

いる．この場合，図 14(b) に示すように，MPI プロセスのロールバックリカバリに使用するチェックポイントのタイミングを 1 つさかのぼる．図 14(b) では，タイミングを 1 つさかのぼったチェックポイントはチェックポイント 5 である．チェックポイントのタイミングをさかのぼってもチェックポイントを見つけることができない場合，さらにさかのぼる．全てのチェックポイントを探し終えるか，利用可能なチェックポイントサーバーがなくなると MPI プロセスのロールバックリカバリを諦める．

次に複数のチェックポイントサーバーから利用可能なチェックポイントの中から最新のチェックポイントを探すために MPICH-Vcl に対して行ったチェックポイントサーバー切替方式の実装を説明する．利用可能なチェックポイントの中から最新のチェックポイントを探索する処理は通信デーモンが行う．通信デーモンを以下に示すように変更した．

通信デーモンにおけるロールバックリカバリの処理を以下に示す．

1. 通信デーモンはロールバックリカバリが開始されるとチェックポイントサーバーに接続する．

2. 通信デーモンとチェックポイントサーバーの接続は MPI プロセスのロールバックリカバリが完了するまで維持される．

3. ロールバックリカバリが完了すると通信デーモンはチェックポイントサーバーとの接続を断つ．

ロールバックリカバリが開始され，通信デーモンがチェックポイントサーバーと接続する部分を，図 15 のフローチャートに示した処理に変更した．

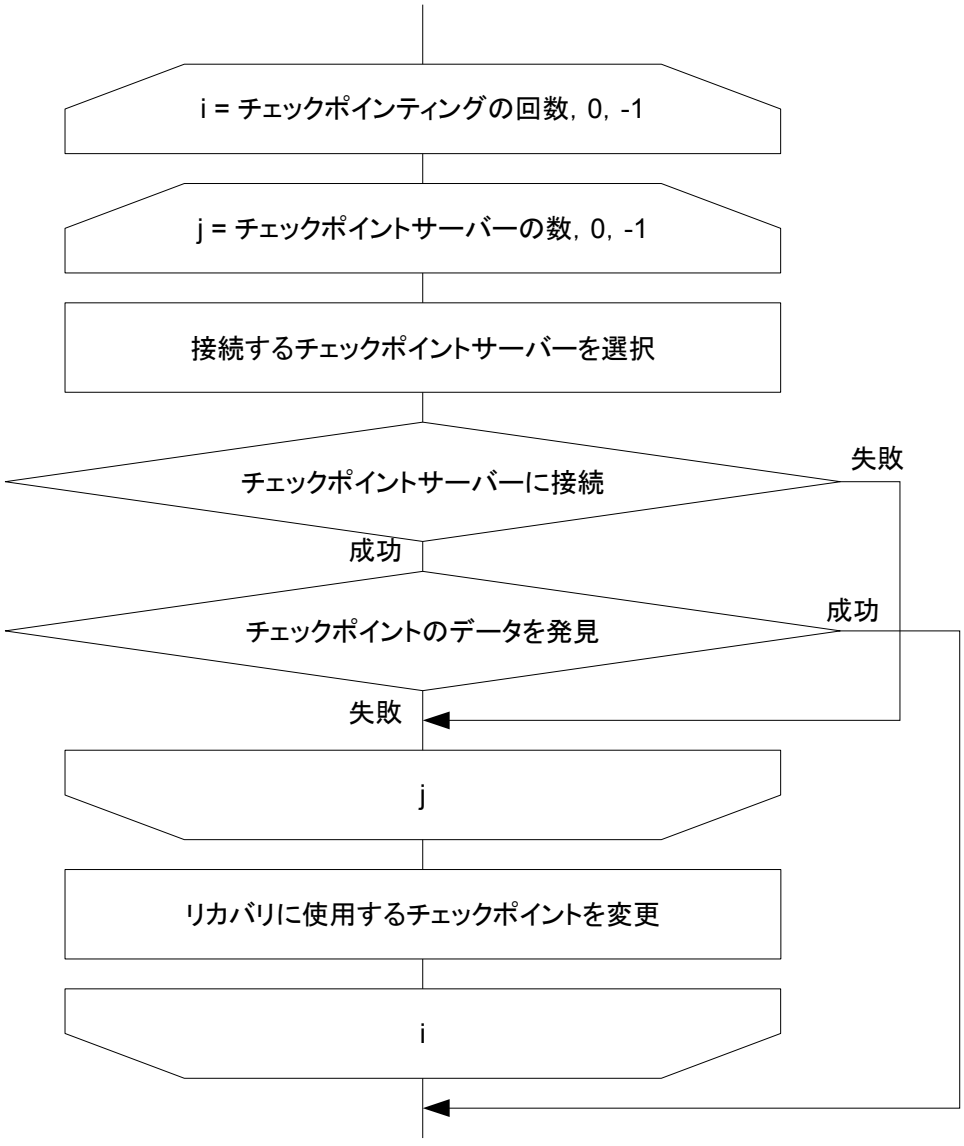


図 15: ロールバックリカバリ時に最新のチェックポイントを探索する処理

8 評価

チェックポイントサーバー冗長化の実装を行ったMPIライブラリを評価した．本章では評価の内容と結果をまとめる．

8.1 動作検証

チェックポインティングとロールバックリカバリにおいて，期待される処理が行われていることを確認した．以下に評価方法と結果を述べる．

動作検証を行った環境を表6に示す．

表 6: 提案方式の動作検証を行った環境

CPU	Intel(R) Pentium(R) III 1400MHz (2/node)
Memory	1GByte
OS	Red Hat Linux release 9
Kernel	2.4.20-8smp
MPICH	mpich-1.2.5.2
ノード数	4

チェックポイントサーバー切替えの動作検証

動作検証の方法 NAS Parallel Benchmark LU Class=A プロセス数=4 を実行し，各チェックポイントサーバーに保存されていたチェックポイントを確認する．

動作検証の結果 動作検証の結果，MPIアプリケーションの実行中にチェックポインティングの失敗によるエラー情報は出力されず，MPIアプリケーションの実行は最後まで行われた．また，各ノードに保存されているチェックポイントを調査し，チェックポインティング毎にチェックポイントサーバーを切替えていることを確認した．

故障が発生した場合の対応の検証

評価方法 NAS Parallel Benchmark LU Class=B プロセス数=4 を実行中に kill コマンドでプロセスを停止させ，MPIアプリケーションの実行に対する影響を調査した．以下に示す3種類の故障を発生させた．

- MPI プロセスのみを停止.
- チェックポイントサーバーのみを停止.
- MPI プロセスと最新のチェックポイントを保存したチェックポイントサーバーを同時に停止.

評価結果 先に述べた故障を発生させたところ，いずれの場合も MPI アプリケーションの実行を続けることができた．また，MPICH-Vcl はチェックポイントサーバーが停止してしまうと MPI アプリケーションの実行を続けることができなかった．従って，チェックポイントサーバーを冗長化することによりチェックポイントサーバーの故障に対応可能となり，オリジナルの MPICH-Vcl よりも耐故障性をあげることができた．以下に詳細を述べる．

MPI プロセスのみの停止 この故障に対してはオリジナルの MPICH-Vcl も対応可能である．チェックポイントサーバー切替方式の実装を行った場合も故障発生時に MPI プロセスをロールバックリカバリし，MPI アプリケーションの実行を続けることを確認した．

チェックポイントサーバーの停止 この故障が発生した場合，停止したチェックポイントサーバーを使わずにチェックポインティングを行い，MPI アプリケーションの実行を続けることを確認した．オリジナルの MPICH-Vcl ではチェックポイントサーバーが停止した場合，MPI アプリケーションの実行を続けることはできない．

MPI プロセスと最新のチェックポイントを保存したチェックポイントサーバーが同時に停止 この故障が発生した場合，最新ではないチェックポイントから MPI プロセスをロールバックリカバリし，MPI アプリケーションの実行を続けることができた．オリジナルの MPICH-Vcl では MPI プロセスとチェックポイントサーバーが同時に故障した場合，ロールバックリカバリ時に通信デーモンがチェックポイントサーバーに対して接続を試みた際，接続が確立できず，MPI プロセスをロールバックリカバリできない．

8.2 オーバーヘッドの測定

提案方式，MPICH-Vcl，MPICH-Vcl（チェックポインティング無し）の 3 つのパターンで NAS Parallel Benchmark の実行時間を測定し，チェックポインティングに

かかるオーバーヘッドを調査した．調査の結果，チェックポインティングのオーバーヘッドはMPICH-Vclと同程度であることを確認した．

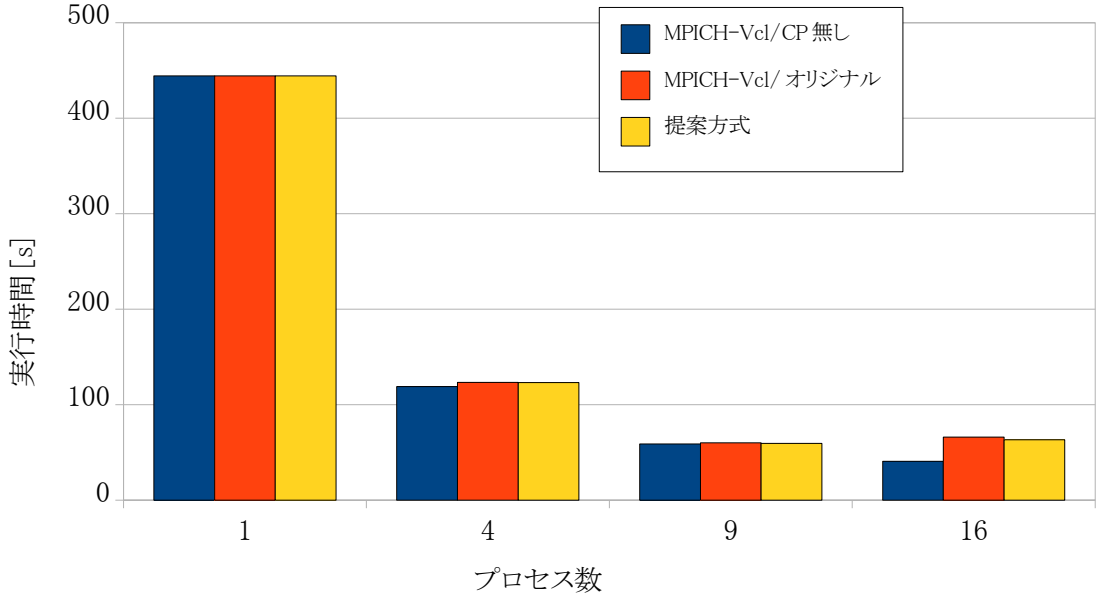
提案方式，MPICH-Vcl，MPICH-Vcl（チェックポインティング無し）の3つのパターンでNAS Parallel Benchmarkの実行時間を測定した．提案方式，MPICH-Vcl，MPICH-Vcl（チェックポインティング無し）のそれぞれの結果を比較した．

評価環境を表7に示す．

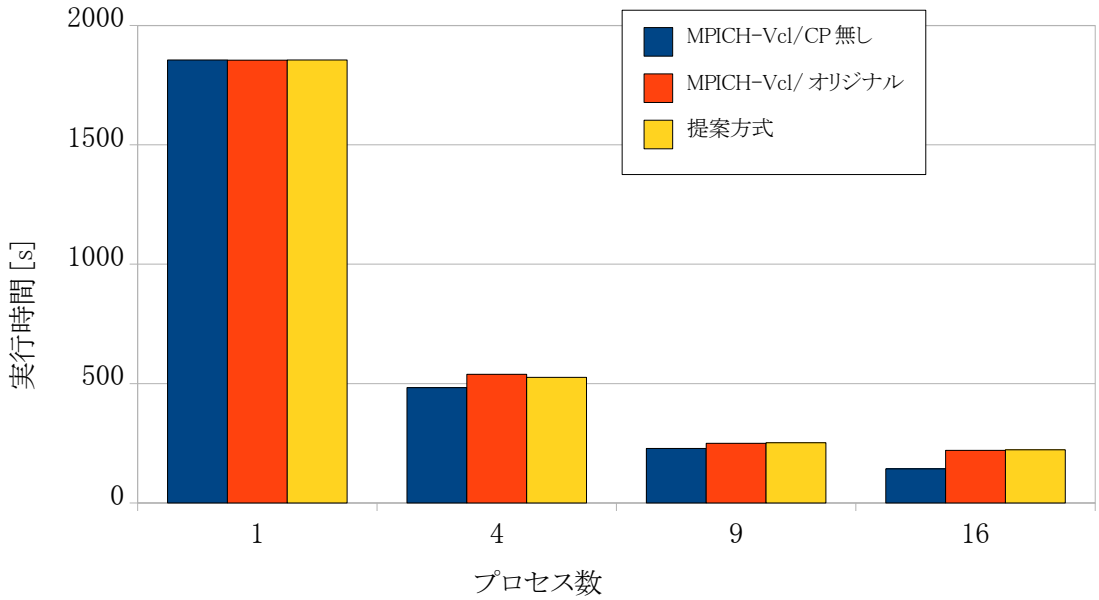
表 7: 提案方式の実行性能評価を行った環境

CPU	Intel(R) Xeon(TM) CPU 2.80GHz (2/node)
Memory	1GByte
OS	CentOS release 4.7
Kernel	Linux 2.6.9
MPICH	mpich-1.2.5.2
ノード数	16

実行時間の測定結果を図16，図17，図18，図19，図20，図21に示す．測定結果から，提案方式のチェックポインティングにかかるオーバーヘッドはMPICH-Vclと同程度であることが確認できた．

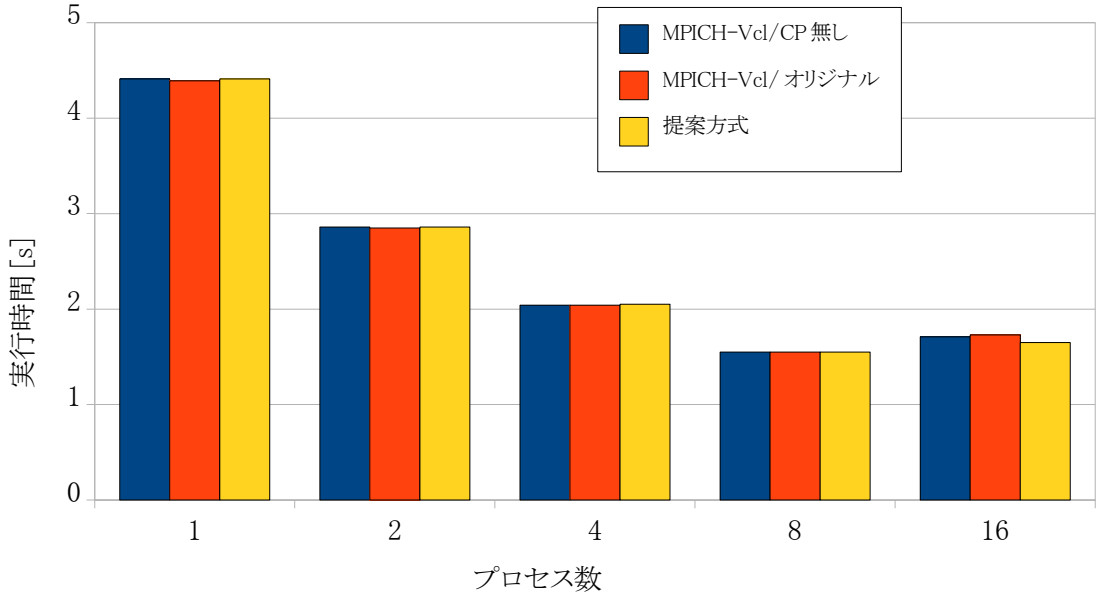


(a) NAS Parallel Benchmark BT Class=A

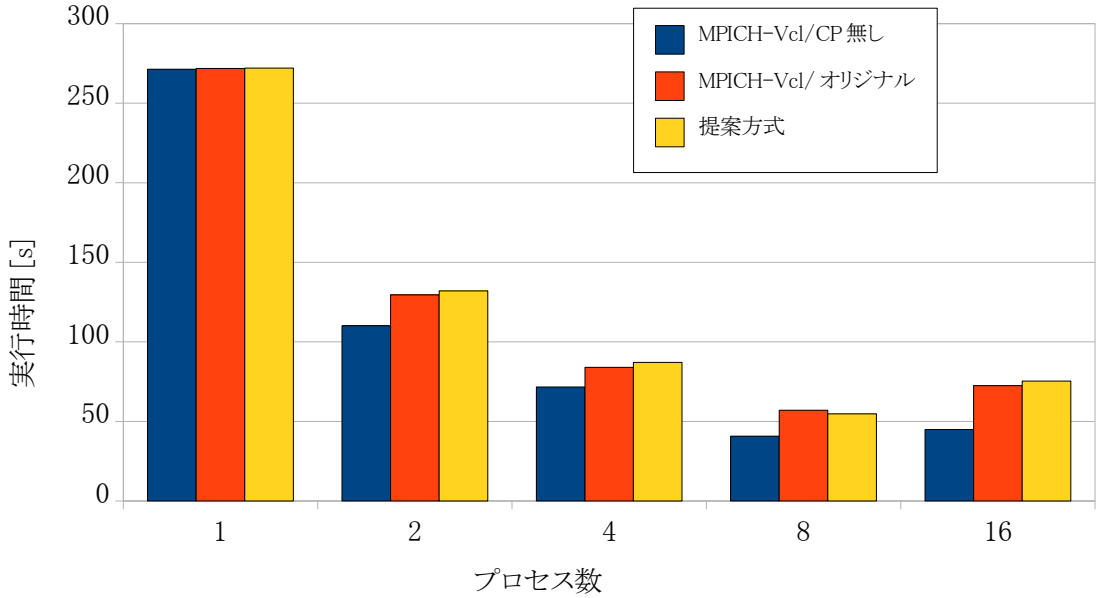


(b) NAS Parallel Benchmark BT Class=B

図 16: 提案方式，MPICH-Vcl，MPICH-Vcl(チェックポインティング無) における NAS Parallel Benchmark BT の実行時間の比較



(a) NAS Parallel Benchmark CG Class=A



(b) NAS Parallel Benchmark CG Class=B

図 17: 提案方式，MPICH-Vcl，MPICH-Vcl(チェックポインティング無) における NAS Parallel Benchmark CG の実行時間の比較

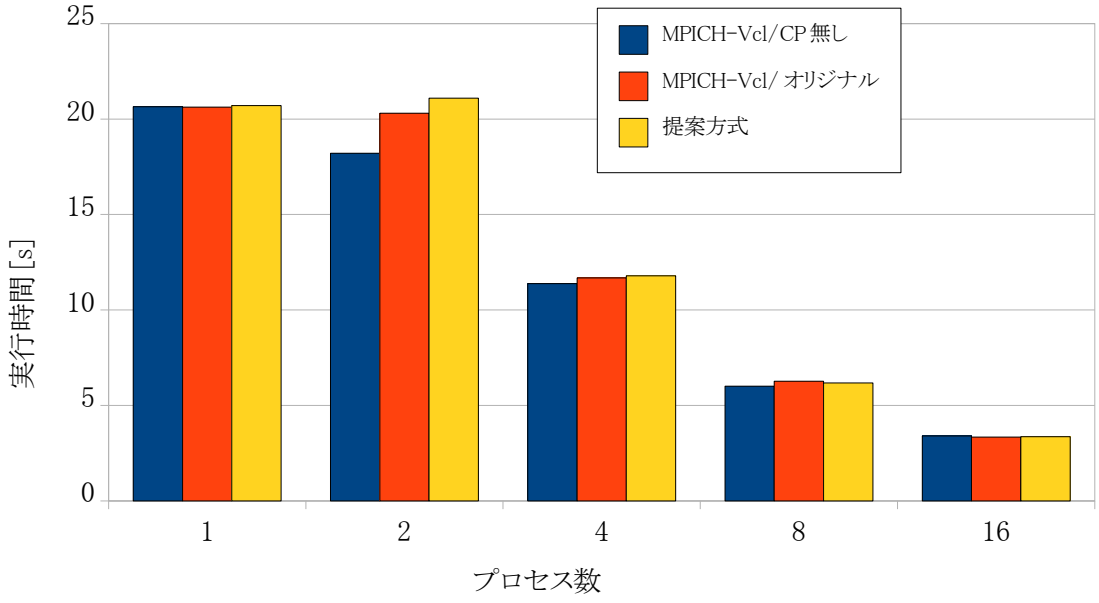
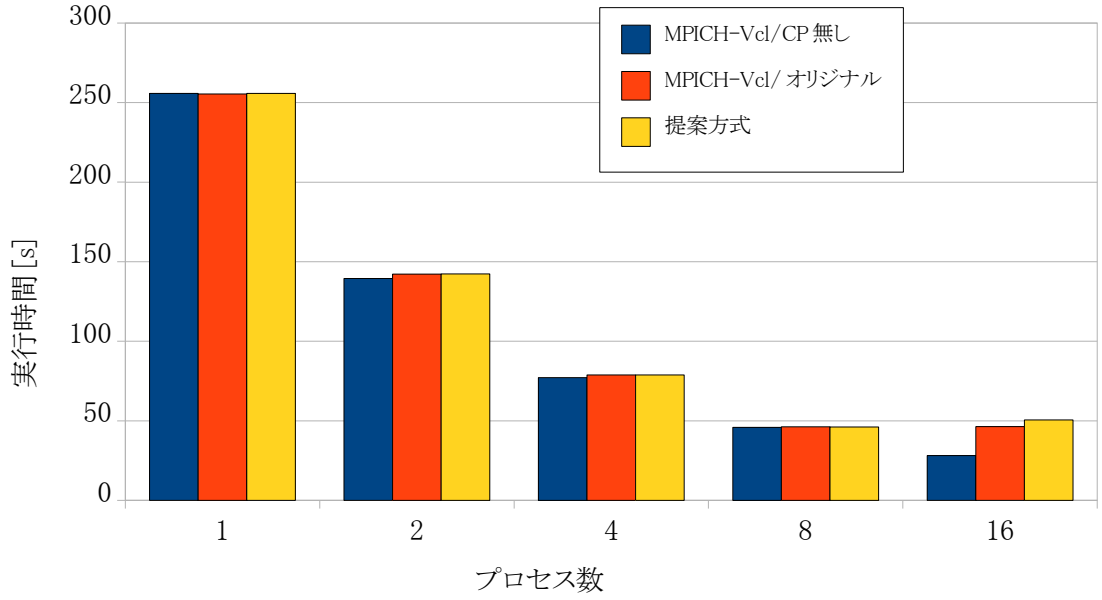
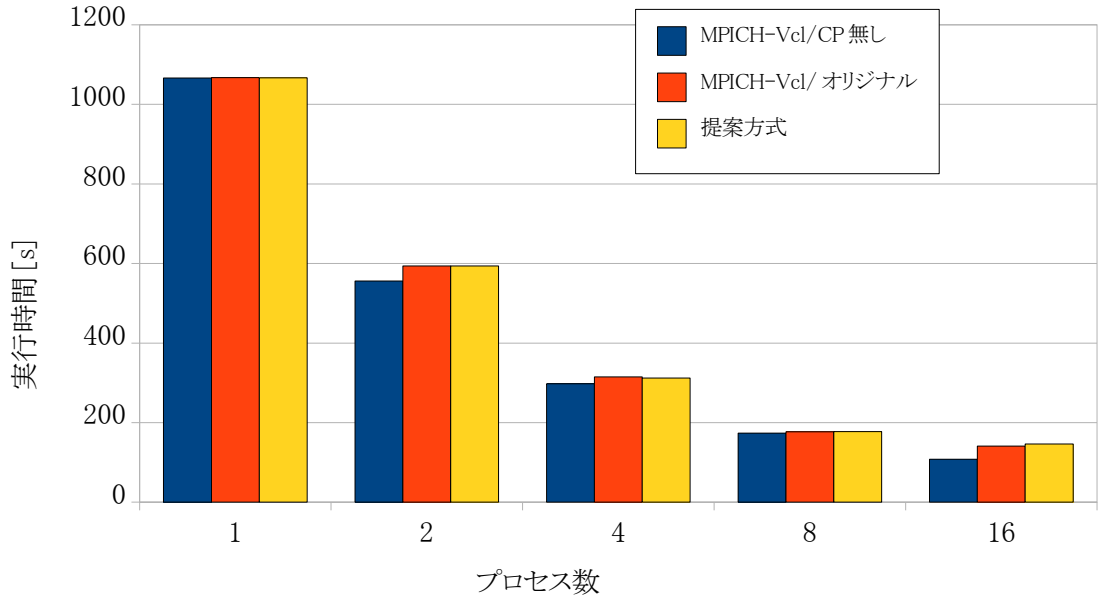


図 18: 提案方式，MPICH-Vcl，MPICH-Vcl(チェックポインティング無) における NAS Parallel Benchmark FT CLASS=A の実行時間の比較

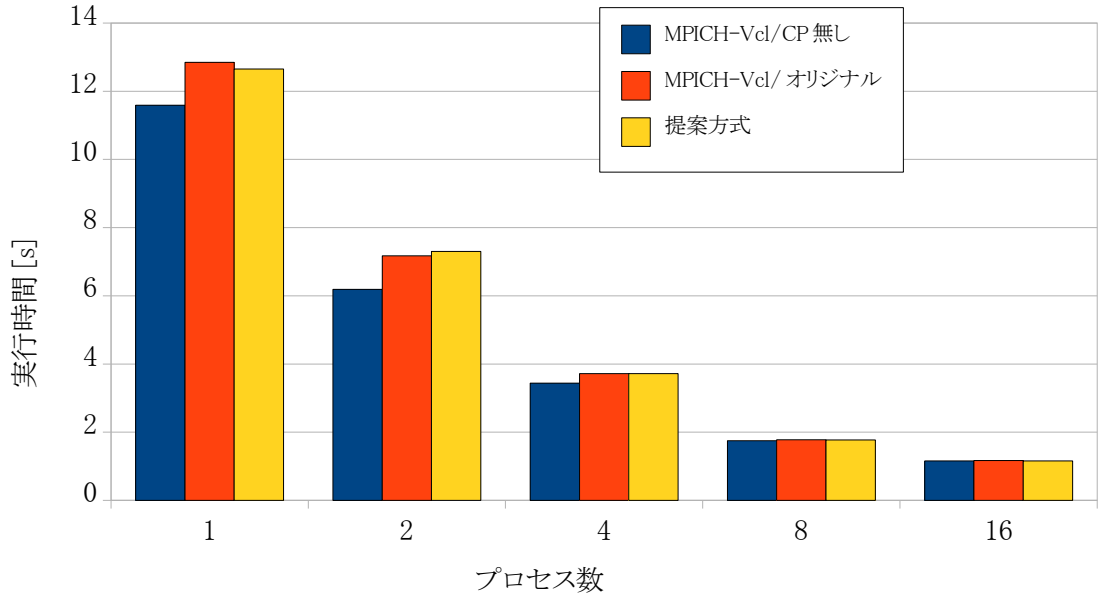


(a) NAS Parallel Benchmark LU Class=A

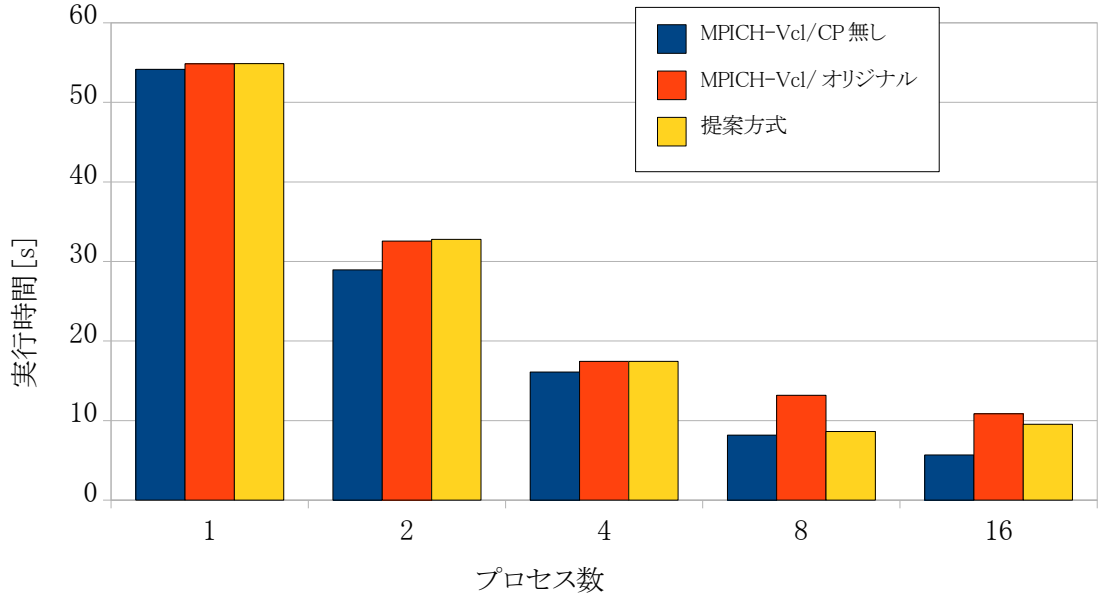


(b) NAS Parallel Benchmark LU Class=B

図 19: 提案方式，MPICH-Vcl，MPICH-Vcl(チェックポインティング無) における NAS Parallel Benchmark LU の実行時間の比較

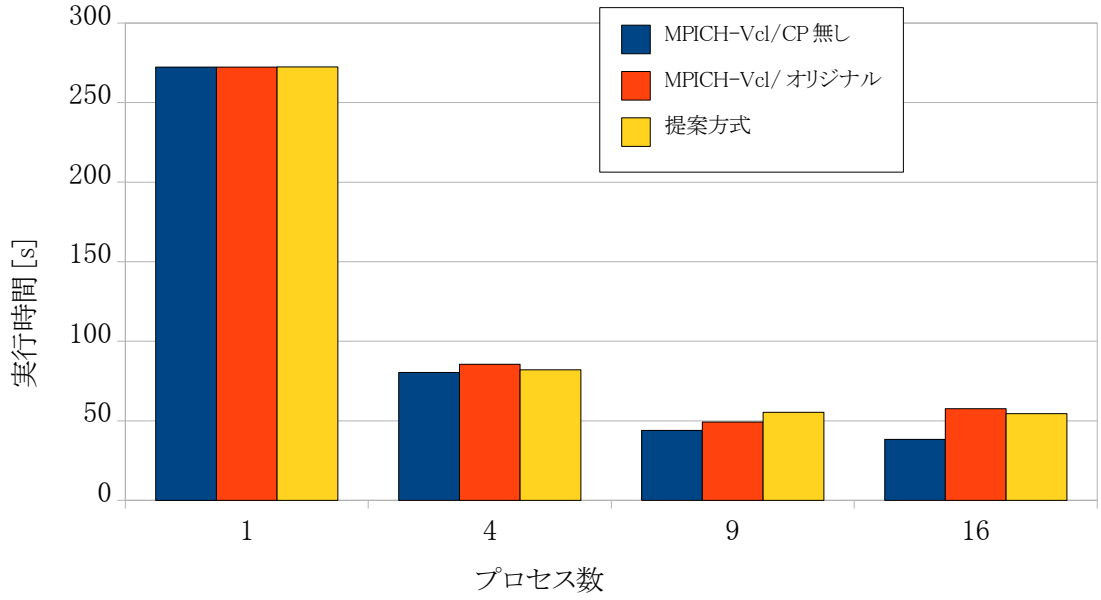


(a) NAS Parallel Benchmark MG Class=A

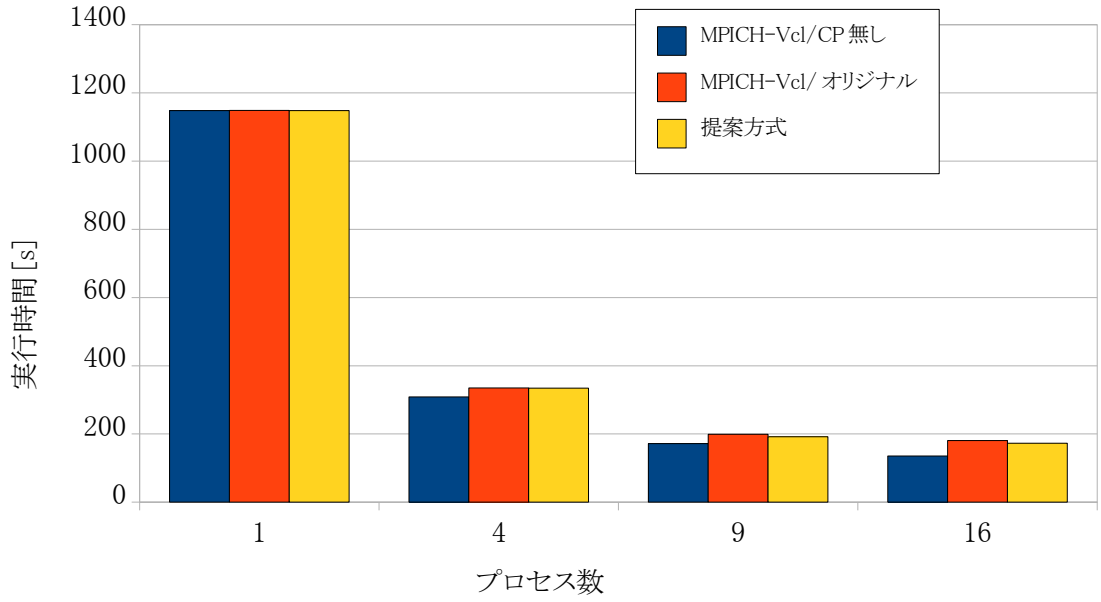


(b) NAS Parallel Benchmark MG Class=B

図 20: 提案方式，MPICH-Vcl，MPICH-Vcl(チェックポインティング無) における NAS Parallel Benchmark MG の実行時間の比較



(a) NAS Parallel Benchmark SP Class=A



(b) NAS Parallel Benchmark SP Class=B

図 21: 提案方式，MPICH-Vcl，MPICH-Vcl(チェックポインティング無) における NAS Parallel Benchmark SP の実行時間の比較

9 関連研究

MPIやコンピュータクラスタにおける耐故障性に関する研究は多く行われている．

MPIに対する耐故障性提供方式 Cocheck[6]，MPI/FT[8]，Parakeet[24]は，MPIアプリケーションに対して耐故障性を付加するMPIライブラリにおける研究である．これらの研究はMPIアプリケーションに対する耐故障性の提供方式が研究対象である．本研究とは耐故障MPIライブラリに関する研究としては共通しているが，MPIアプリケーションに対して耐故障性を付加するプロセスにおける耐故障を研究対象としている点において，これらの研究と異なる．

コンピュータクラスタ管理プロセスにおける耐故障方式 文献[25]では，コンピュータクラスタの管理を行うプロセスにおける耐故障方式を提案している．他のプロセスに対して耐故障性を提供するプロセスにおける耐故障方式に関する研究として本研究と共通している．しかし，本研究の対象はMPIアプリケーションのチェックポイントを管理するプロセスにおける耐故障である点において，この研究とは異なる．

チェックポイントにおける耐故障方式に関する研究 チェックポイントにおける耐故障式としては，本論文で取り上げたSkewed Checkpointing[18, 19]，MIR[16]，CFS[16]，2-level Recovery Scheme[17]が提案されている．

Skewed CheckpointingとMIRはチェックポイントを複数のノードに保存するミラーリングを行うことで，チェックポイントの冗長性を確保している．

CFSは高信頼なノードにチェックポイントを保存することで，チェックポイントの信頼性を向上させている．MIRよりも信頼性が高い半面，チェックポインティング時，ロールバックリカバリ時に高信頼なノードにアクセスが集中するためオーバーヘッドが大きい．

2-level Recovery SchemeはCFSとMIRの2つの方式を組み合わせた耐故障方式である．CFSによりチェックポイントの信頼性を向上させることに加え，MIRを行うことでCFSを用いる頻度を減らし，チェックポインティングにかかるオーバーヘッドを軽減している．

このほかにも，パリティを用いてチェックポイントを冗長化する方式が提案されている．パリティを用いる方式としてはChecksum-Based Encoding[27, 28]とWeighted-Checksum-Based Encoding[27, 29]が知られている．パリティを用いる方式を実装したコンピュータクラスタのシステムソフトウェアとしては

SCore[30, 31, 32] が知られている．本研究ではパリティを用いたチェックポイントの高信頼化は行っていない．

Checksum-Based Encoding は Coordinated Checkpointing を対象としている．作成したチェックポイントは各ノードのローカルメモリに保存される．また，各ノードのチェックポイントはパリティを付加し，1 つの Checksum data にまとめ，専用ノードに保存される．実行中のプロセスに故障が発生した場合，故障が発生していないプロセスはローカルに保存されたチェックポイントを用いてロールバックリカバリされる．故障が発生したプロセスは Checksum data から再構築したチェックポイントを用いて，予備ノード上でロールバックリカバリされる．単一プロセスの故障からロールバックリカバリできる．

Weighted-Checksum-Based Encoding は Checksum-Based Encoding の拡張である．Checksum data を複数作成することで複数のプロセス故障からロールバックリカバリを可能にしている．

SCore ではチェックポイントを各ノードのローカルディスクに保存する．また，チェックポイントは複数のデータに分割した上でパリティ演算を行い，他ノードに分散して保存することでチェックポイントの冗長性を確保している．ノードの永久故障や通信障害によってローカルディスクに保存されたチェックポイントが失われた場合はパリティからチェックポイントを復元し，故障したプロセスのロールバックリカバリに用いる．

10 おわりに

10.1 まとめ

本研究ではMPICH-Vclに対してチェックポイントサーバーの冗長化を行った．チェックポイントサーバーを冗長化する手法としてチェックポイントサーバー切替方式を実装した．

チェックポイントサーバー切替方式を実装したMPIライブラリに対して，故障を発生させ，動作検証を行った．動作検証の結果，提案方式がチェックポイントサーバーの故障に対応可能であることを確認した．

また，MPICH-Vcl（チェックポインティング無），MPICH-Vcl（チェックポインティング有），チェックポイントサーバー切替方式のMPIアプリケーション実行時間を，NAS Parallel Benchmarkを実行し，実行時間を比較した．その結果，提案方式のチェックポインティングによるオーバーヘッドは，MPICH-Vclと同程度であることを確認した．

10.2 今後の課題

今後の課題としては以下の2つが考えられる．

チェックポイントサーバー以外のプロセスにおける耐故障

今回はチェックポイントサーバーにおける故障の対策としてチェックポイントサーバーの冗長化を行った．チェックポイントサーバー以外のプロセスに故障が発生した場合もMPIアプリケーションの耐故障性や実行に悪影響を与えるため，MPICH-Vclの他のプロセスに故障が発生した場合についても対応可能にする必要がある．

他のチェックポイントサーバーにおける耐故障方式

今回はチェックポイントサーバー冗長化の手法として，チェックポイントサーバー切替方式を実装した．Skewed CheckpointingやMIRなどの手法を実装した場合，チェックポイントサーバー切替方式とは対応可能な故障やMPIアプリケーションの実行性能が異なることが予想される．従って，他のチェックポイントサーバー冗長化の手法についても実装を行い，対応可能な故障や，MPIアプリケーションの実行性能などの特性を調査することは課題である．

謝辞

本研究を進めるにあたり，多大なるご指導，ご助言を下さいました高性能コンピューティング学講座本多弘樹教授，近藤正章准教授，平澤将一助教，ならびに渡邊啓正氏，田邊浩志氏に心より感謝いたします．また，研究を進める上でご支援やご助言を頂きました高性能コンピューティング学講座の皆々様に深く感謝いたします．

参考文献

[1] P. パチェコ 著, 秋葉 博 訳. ‘‘ MPI 並列プログラミング ’’. 培風社, (2001) .

[2] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, Timothy S. Woodall. “Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation”. *Lecture Notes in Computer Science*, Vol.3241, 2004, pp97-104.

[3] Jeffrey M. Squyres, Andrew Lumsdaine. “The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing”, *International Journal of High Performance Computing Application*, Venice, Italy, Sept. 2003. Springer.

[4] William Gropp, Ewing Lusk, Nathan Doss, Anthony Skjellum. “A high-performance, prtable implementation of the MPI message passing interface standard” *Parallel Computing*, 22(6):789-828, September 1996.

[5] Jiuxing Liu, Jiesheng Wu, Sushmitha P. Kini, Peter Wyckoff, and Dhabaleswar K. Panda. “High Performance RDMA-Based MPI Implementation over InfiniBand”. in *the Proceedings of 17th Annual ACM International Conference on Supercomputing*. San Francisco Bay Area. June, 2003.

[6] G. Stellner. “CoCheck: Checkpointing and Process Migration for MPI”. *In Proceedings of the 10th International Parallel Processing Symposium*, Honolulu, HI, 1996.

[7] Adnan Agbaria1, Roy Friedman. “Starfish: Fault-Tolerant Dynamic MPI Programs on Clusters of Workstations”. *Cluster Computing*, Vol.6, No.3, 1999, pp.227-236.

[8] Rajanikanth Batchu, Yoginder S. Dandass, Anthony Skjellum, Murali Beddhu. MPI/FT: A Model-Based Approach to Low-Overhead Fault Tolerant Message-Passing Middleware *Parallel Processing Letters*, Vol.10, December 2000, pp.371-382.

[9] E.N.(Moortaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, David B.Johnson. “A Survey of Rollback-Recovery Protocols in Message-Passing Systems”. *ACM Computing Surveys*, Vol.34, No.3, September 2002, pp.375-408.

[10] D. L. Russell. State restoration in systems of communicating processes. Technical report, IEEE Transactions on Software Engineering, SF6:2, 3 1980.

[11] Aurélien Bouteiller, Pierre Lemarinier, Géraud Krawezik, Franck Cappello. “Coordinated checkpointing versus message log for fault tolerant MPI”. in *proceedings of The 2003 IEEE International Conference on Cluster Computing*, 2003, pp.242-250.

[12] Camille Coti, Thomas Herault, Pierre Lemarinier, Laurence Pilard, Ala Rezmerita, Eric Rodriguez, Franck Cappello. “Blocking vs. Non-Blocking Coordinated Checkpointing for Large-Scale Fault Tolerant MPI”. in *proceedings of The 2006 IEEE/ACM Conference on Supercomputing*, 2006, pp.18.

[13] Darius Buntinas, Camille Coti, Thomas Heralut, Pierre Lemarinier, Laurence Pilard, Ala Rezmerita, Eric Rodriguez, Franck Cappello. “Blocking vs. Non-Blocking Coordinated Checkpointing for Large-Scale Fault Tolerant MPI”. *Future Generation Computer Systemm, Elsevier*. 2007.

[14] Pierre Lemarinier, Aurelien Bouteiller, Thomas Herault, Geraud Krawezik, Franck Cappello. “Improved Message logging versus Improved coordinated checkpointing for fault tolerant MPI”. in *proceedings of The 2004 IEEE International Conference on Cluster Computing*. 2004, pp.115-124.

[15] Aurélien Bouteiller, Thomas Herault, Géraud Krawezik, Pierre Lemarinier, Franck Cappello. “MPICH-V Project: a Multiprotocol Automatic Fault Tolerant MPI”. in *International Journal of High Performance Computing and Applications*. Vol.20, No.3, 2006, pp.319-333.

[16] James S. Plank. “Improving the Performance fo Coordinated Checkpointers on Nerworks of Workstations using RAID Techniques”. in *proceedings of the 15th Symposium on Reliable Distributed Systems(SRDS ’96)*, 1996, pp.76-85.

[17] Nitin H.Vaidya. “A Case for Two-Level Recovery Schemes”. *IEEE Transactions on Computers*, Vol.47, No.6, pp656-666, 1998.

[18] Hiroshi NAKAMURA, Rakuro HAYASHIDA, Masaaki KONDO, Yuya YAJIMA, Masashi IMAI, Takashi NANYA. “Skewed Checkpointing for Tolerateing Multi-Node Failures”. *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems*, 2004, pp.116-125.

[19] 南谷崇，中村宏. “クラスタシステムの高信頼化技術”. 21 世紀 COE プログラム 情報科学技術戦略コア 大域ディペンダブル情報基盤プロジェクト報告, 2003.

[20] Aurélien Bouteiller, Thomas Herault, Géraud Krawezik, Pierre Lemarinier, Franck Cappello. “Impact of Event Logger on Causal Message Logging Protocols for Fault Tolerant MPI”. in *proceedings of 19th IEEE International Parallel and Distributed Processing Symposium(IPDPS'05) - Papers*, 2005, pp.97.

[21] William Hoarau, Pierre Lemarinier, Thomas Herault, Eric Rodriguez, Sébastien Tixeuil, Franck Cappello. “FAIL-MPI: How Fault-Tolerant Is Fault-Tolerant MPI?”. in *proceedings of The 2006 IEEE International Conference on Cluster Computing*, 2006, pp.1-10.

[22] Aurélien Bouteiller, Franck Cappello, Thomas Hérault, Géraud Krawezik, Pierre Lemarinier, Frédéric Magniette. “MPICH-V2: a Fault Tolerant MPI for Volatile Nodes based on Pessimistic Sender Based Message Logging”. in *proceedings of The 2003 IEEE/ACM Conference on Supercomputing*. 2003, pp.25.

[23] George Bosilca, Aurélien Bouteiller, Franck Cappello, Samir Djilali, Gilles Fédak, Cécile Germain, Thomas Hérault, Pierre Lemarinier, Oleg Lodygensky, Frédéric Magnierre, Vincent Néri, Anton Selikhov. “MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes”. in *proceedings of The 2002 IEEE/ACM Conference on Supercomputing*. 2002, pp.29.

[24] 高宮 安仁，松岡 聡. ‘‘ ユーザ透過な耐故障性を実現する MPI に向けて ’’. 情報処理学会研究報告，Vol.2001，No.77，2001，pp.129-134.

[25] Ming Li, Daniel Goldberg, Wenchao Tao, Yuval Tamir. “Fault-Tolerant Cluster Management for Reliable High-Performance Computing”, *Proceedings of International Conference on Parallel and Distributed Computing and Systems*,

[26] Rob F. Van der Wijngaart. “NAS Parallel Benchmarks version 2.4.”. Technical Report NAS Technical Report NAS-02-007, NASA Advanced Supercomputing Division, NASA Ames Research Center, October 2002.

[27] Zizhong Chen, Graham E. Fagg, Edgar Gabriel, Julien Langou, Thara Angskun, George Bosilca, and Jack Dongarra. “Fault tolerant high performance computing by a coding approach”. *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, 2005, pp.213-223.

[28] J. S. Plank, and K. Li. “Faster checkpointing with n+1 parity”. In *FTCS*, 1994, p.288-297.

[29] J. S. Plank. “A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems”. *Software Practice & Experience*, 27(9):995-1012, 1997.

[30] Yutaka Ishikawa, Hiroshi Tezuka, Atsushi Hori, Shinji Sumimoto, Toshiyuki Takahashi, Francis O’Carroll, Hiroshi Harada, “RWC PC cluster II and SCore cluster system software - High performance Linux cluster”. Proc. 5th Annual Linux Expo, 1999, pp.55-62.

[31] Masaaki Kondo, Takuro Hayashida, Masashi Imai, Hiroshi Nakamura, Takashi Nanya, Atsushi Hori. “Evaluation of Checkpointing Mechanism on SCore Cluster System”. IEICE, Vol.E86-D, No.12, 2003, pp. 2553-2562

[32] 南谷崇，中村宏. “大域ディペンダブルプロジェクト〜ディペンダブルアーキテクチャグループ 南谷・中村研究室〜”. 大域ディペンダブル情報基盤プロジェクト報告，平成 15 年.